LAB GUIDE

Introduction to API Management

Overview

In this lab, you will create a Web API using ASP.NET MVC that will then be deployed into Azure API Apps. You will also integrate Swagger using the Swashbuckle NuGet package to automatically generate usage documentation for the Web API. From there you will setup a new API Management Service within Azure, and publish a custom Web API deployed to an Azure API App to be a Managed API.

Time Estimate

* 1 Hour 45 minutes

Exercise 1: Deploying the Lab Environment

Overview

In this exercise, you will start the Real Time Lab Environment for the lab. This lab will provision a virtual machine with the software necessary to complete the hands-on lab. You will also have access to a Microsoft Azure subscription to complete the tasks. Do not create more resources in Azure then the lab requires or your lab may be terminated.

Time Estimate

* 10-15 minutes

Task 1: Login to the Lab Environment

1. On the View Lab Environment tab, click the Connect button by the LABVM and login with the credentials on the same tab.
2. From within the LABVM, launch Chrome and navigate to https://portal.azure.com. Login using the Azure Credentials from the View Lab Environment tab of the SkillMeUp.com portal.

**Lab Tips**

1. After each exercise is completed, click the Mark Complete check mark to track your progress. After all exercises are complete you can view your completion certificates under my account.
2. After you have completed all of the lab exercises, click the End Lab button.

Summary

In this exercise, you successfully connected to the lab VM and, from there, accessed the Azure Management Portal in Google Chrome using the provided credentials.

LAB 2

Exercise 2: Create and Deploy an API App to Azure App Service

Overview

In this exercise, you will create a Web API using ASP.NET MVC that will then be deployed into Azure API Apps. You will also integrate Swagger using the Swashbuckle NuGet package to automatically generate usage documentation for the Web API.

Time Estimate

* 45 minutes

Task 1: Create Resource Group and Storage Account

In this Task, you will create a Storage Account that will use Table Storage for the back-end data store for the Contact entities the Web API will interact with.

1. Open your browser and navigate to [http://portal.azure.com](http://portal.azure.com/)
2. In the Azure Portal, click on the **+ Create a resource** button in the toolbar, then **Storage**, and click on **Storage account**
3. Enter the following values into the **Create storage account** blade:
   1. Name: **enter an unique name**
   2. Replication: **Locally-redundant storage (LRS)**
   3. Resource group: **LABWEBAPI**
   4. Location: **choose the Azure region closest to you**
4. Click **Create**
5. Once the **Resource Group** and **Storage Account** have been created, click on **Resource groups**, click the **LABWEBAPI** resource group, then click on the **Storage Account** that was just created.
6. On the **Storage Account** blade, click on **Access keys**
7. Copy the **Connection String** for the Storage Account and save it for use later.

Task 2: Create Web API Project

In this exercise, you will create a new Web API project within Visual Studio that is configured to deploy to Azure App Service.

1. Open **Visual Studio**
2. Click on the **File** menu, then **New**, and **Project…**
3. On the **New Project** dialog, select the **ASP.NET Web Application (.NET Framework)** project template.
4. Enter a name **LabWebApi** for the project name, and click **OK**.
5. On the **New ASP.NET Web Application** dialog, click on **Web API**, and click **OK**
6. Once the project has been created, right-click on your project within the Solution Explorer window and then **References**, and then click on **Manage NuGet Packages…**
7. In the **NuGet** **References** window , click on **Browse** and type **WindowsAzure.Storage** into the search box.
8. Click on the **WindowsAzure.Storage** NuGet package, then click the **Install** button to add a reference to it in the project.
9. Click **OK** on the **Review Changes** dialog
10. Click **I Accept** on the **License Acceptance** dialog. After installation completes, save all and re-build the project.
11. Then, click on **Updates** of the Nuget Package Manager and install all available updates. Do this until there are no more updates available. Then, save all files and re-build the project again.
12. Within the **Solution Explorer** window in the **LabWebApi** project, expand the **Controllers** folder.
13. Delete the **ValuesController.cs** file from the project. This will not be needed in this lab.
14. Right-click on the **Controllers** folder, and click on **Add**, then **Controller…**.
15. In the **Add Scaffold** dialog, select the **Web API 2 Controller – Empty** option and click **Add**.
16. Name the Controller, **ContactsController**, then click **Add**.
17. Copy the contents of the **ContactsController.cs** code file from **C:\OpsgilityTraining** and paste it into the newly created code file within the project.
18. Within the **Solution Explorer** window, open the **Web.config** file.
19. Above the **<appSettings>** element, add a new element.
20. Within the **<connectionStrings>** configuration element, paste in the following snippet to add the **Connection String** for the **Azure Storage Account**. Replace the **{connection-string}** value with the actual **Storage Account Connection String** that was copied in the previous exercise.

<add name="tableStorageConnString" connectionString="{connection-string}" />

1. The Connection String in the **Web.config** file should look similar to the following screenshot:

Task 3: Integrate Swagger

In this Task, you will integrate Swagger into the Web API project using the Swashbuckle NuGet package.

1. With the Web API project still open in Visual Studio, right-click on the project **References** within the **Solution Explorer** window, and click on **Manage NuGet Packages…**
2. In the **NuGet References** window, click on **Browse**, then type **Swashbuckle** into the search box.
3. Click on the **Swashbuckle** NuGet package, then click **Install** to add a reference to it within the project.
4. On the **Review Changes** dialog, click **OK**. After **Swashbuckle** is installed, again, ensure all available package updates are installed and then save all project files and rebuild the solution.
5. Finally, click on the **Start** button (or press the **F5** key) in the Visual Studio Toolbar to run the Web API locally to ensure everything compiles and the app can run. Once running locally, the app will open up in a new browser window.
6. In the browser window that opened, navigate to the **/swagger** folder on the app to view the **Swagger UI**.
7. Close the browser window and stop debugging the app.

Task 4: Deploy to Azure App Service API App

In this Task, you will deploy the newly build Web API app out to Azure App Service.

1. Open a browser window or new tab and navigate to the Azure Portal at [http://portal.azure.com](http://portal.azure.com/)
2. Click on the **+ Create a resource** button in the navigation.
3. Type **API App** into the **Search the marketplace** box and press **Enter**.
4. Click on the **API App** option in the search results.
5. Click **Create**.
6. On the **API App** blade, enter the following values:
   * App name: **Enter a unique app name**
   * Resource Group: Choose the existing **LABWEBAPI** resource group
7. Click on **App Service plan/Location**, then **Create New**.
8. In the **New App Service Plan** blade, enter an **App Service plan name**, and choose the same **Location** chosen for the Storage Account that was previously created.
9. Click on **Pricing tier**, then, on the next screen under **Production**, choose the **S1 Standard** pricing tier, and click **Select**.
10. Click **OK**.
11. Click **Create**.
12. Open the **LabWebApi** project in Visual Studio, right-click the **Project** within the **Solution Explorer**window, then click **Publish…**.
13. On the **Publish** window, click the **Microsoft Azure App Service** option, select the **Select Existing**radio button, then click **Publish**.
14. If not logged into your Azure Subscription, click on the button in the upper-right of the **App Service**dialog, click **Add an account…** and follow the prompt to log into your Azure Subscription.
15. Expand the **LABWEBAPI** Resource Group and select the **API App** that was just created through the Azure Portal, then click **OK**.
16. Once the publishing completes, Visual Studio will open a **Browse** tab navigated to the API App running in Azure.
17. The Web Deploy Publish status will be shown in the **Output** window.

Task 5: Test out the API

In this Task, you will test out the Web API by making calls against it using the Swagger UI within a web browser.

1. Open a new browser window and navigate to the domain name for the Azure API App.
2. Open the **Swagger UI** for the API App by navigating to **http://<yourappname>.azurewebsites.net/swagger**.
3. Click on **Contacts** to expand out the different **API Actions** available.
4. Click on and expand the **PUT /api/Contacts** action that can be used to **Insert** a new contact record.
5. Enter the following **JSON** snippet into the field for the **value** parameter, and click **Try it out!**

{

"Age": 42,

"PrimaryPhone": "555-1234",

"Address1": "123 My Street",

"Address2": "",

"City": "Localville",

"StateProv": "NY",

"PostalCode": "11111",

"PartitionKey": "Jones",

"RowKey": "Martha",

"Timestamp": "2016-11-23T20:12:48.587Z",

"ETag": "string"

}

1. Scroll down to see the **Response Code** of **204** meaning the action executed successfully and didn’t return any content in the response. Conversely, a **500** here would signify a Server Error.
2. Click on and expand the **GET /api/Contacts** action that can be used to list all the Contacts stored, and click **Try it out!**
3. Scroll down and view the **Response Body** to see the **JSON** data returned by the API action method, as well as the **Response Code** of **200**.

Task 6: Register Swagger API Definition with API App

In this Task, you will log into the Azure Portal, and setup the API Definition property in the API App with the correct location of the Swagger docs API for the Web API.

1. In a new browser window, or tab, navigate to the Azure Portal at [http://portal.azure.com](http://portal.azure.com/)
2. Click on **Resource Groups**, then select the **LABWEBAPI** Resource group, and click on the **API App**.
3. On the **API App** blade, scroll down through the settings options and, under the **API** section, click on **API Definition**.
4. Enter the location to the **Swagger** documentation API Definition into the **API definition location** field. The value to enter here is the full **Domain name of your API App**, with **/swagger/docs/v1** appended to the end of it.
5. Click **Save**

Task 7: Generate Client SDK

In this Task, you will use Swagger and Visual Studio to generate a Client SDK for your Web API that will be used within a .NET Console Application.

1. Open a new instance of **Visual Studio**. Click **File** > **New** > **Project…**.
2. In the **New Project** dialog, navigate to the **Windows Classic Desktop** project template section and select the **Console App** template.
3. Type **ApiCLient** as the name for the Console Application project, then click **OK**.
4. In the **Solution Explorer** window, right-click on the project, then click on **Add** > **REST API Client…**
5. On the **Add REST API Client** dialog, select the **Swagger Url** option and click the **Select Azure Asset…** button. After Visual Studio installs the appropriate NuGet packages, right-click the project folder again and select **Manage NuGet Packages**. Install any newly available updates, then **Save All**and, from the **Build** menu, re-build the solution.
6. On the **App Service** dialog, ensure the button in the upper-right shows that you are logged into your Azure Subscription. If not, then click the button and select **Add an account…** then follow the prompt to log into your Azure Subscription.
7. Expand the **LABWEBAPI** resource group, and select the **API App**.

If the API App doesn’t show in this dialog, then the API Definition Location wasn’t configured in Exercise 1. Configure the definition as per Exercise 1 if this is the case.

1. Click **OK**.
2. Click **OK** on the **Add REST API Client** dialog.
3. Once the **Client SDK** code has been generated, it will be saved into a subfolder within the Project with the same name as the **API App**. You can navigate to this folder to see the full Client SDK source code.
4. Open the **Program.cs** source code file.
5. Add the following line of code to the **Main** method to instantiate a new instance of the **API App Client**. The class name used will be the same name as your API App with the first letter capitalized, so be sure to change the name of the class from shown below:

var apiClient = new <capitalized\_app\_name>(

new Microsoft.Rest.TokenCredentials("test-token")

);

1. Paste in the following line of code to create a new **Contact** object that will be used to insert a new record.

var newContact = new Models.Contact

{

PartitionKey = "Gates",

RowKey = "William",

City = "New York",

StateProv = "NY",

PrimaryPhone = "555-1234"

};

1. Paste in the following line of code that uses the **Put** action method on the **Contacts** **API** to insert a new **Contact** record.

apiClient.Contacts.Put(newContact);

1. Press **Ctrl-F5** to debug the console app and ensure it executes without error.

NOTE: If you have a build error, re-type the name of your api app into the following line and when VS tries to autocorrect, chose the class that matches your api app name.

1. Delete the last two 2 sections of code you inserted, leaving the **apiClient** variable declaration.
2. Paste in the following code that retrieves **All Contacts** using the **Get** action method on the **Contacts API**, then writes them out to the Console.

var allContacts = apiClient.Contacts.Get();

foreach(var contact in allContacts)

{

Console.WriteLine(

string.Format("{0}, {1} - {2}, {3}",

contact.RowKey, // first name

contact.PartitionKey, // last name

contact.City,

contact.StateProv

)

);

}

1. Press **F5** to see the output of all contacts stored by your API in the Console.

Summary

In this exercise, among other things, you created a Web API project in Visual Studio, configured it to connect to your API app in Azure, integrated Swagger into your project, registered the API definition with your API App and generated a client SDK for your API.

LAB-3

Exercise 3: Create Managed API and Published API Product

Overview

In this exercise, you will setup a new API Management Service within Azure, and publish a custom Web API deployed to an Azure API App to be a Managed API.

Time Estimate

* 60 minutes

Task 1: Create API Management Instance

In this Task, you will create a new API Management service within your Azure Subscription.

1. Open your browser and navigate to [http://portal.azure.com](http://portal.azure.com/).
2. In the Azure Portal, click on the **+ Create a resource** button in the left-hand navigation bar, then search for **api management**. Click on the result.
3. On the **API Management service** blade, enter the following values:
   1. Name: **enter a unique name**
   2. Resource group: **LABAPIGATEWAY**
   3. Organization name: **Enter a name for your organization**
   4. Administrator email: **Enter the lab-provided user email address.**
   5. Pricing tier: **Developer (No SLA)**
4. Click **Create**

NOTE: It can take around 30 minutes for the API Management Service to be provisioned. You will get an email when it is fully provisioned.

Task 2: Import API

In this Task, you will import the Web API previously deployed to an Azure API App into the API Management service.

1. Within the Azure Portal, click on **Resource groups**, then the **LABAPIGATEWAY** resource group, then click on the **API Management service**.
2. On the **API Management service** blade, under the **API Management** menu section, click **APIs**.
3. Select the **+ Add API** menu section and then the **API App** option.
4. On the **Create from API App** dialog, click the **Browse** button for the **API App** field and select the app you created earlier. For the **API URL Suffix** field, enter **contacts**. For the **Products** field, click **Starter**. Accept the defaults for the other fields and finally click **Create**.

When the importing completes, you should see the following screen appear:

Task 3: Call API from Developer Portal

In this Task, you will use the API Management Developer Portal to test out the newly imported API.

1. From the **Overview** screen of the **API Management** blade, click on **Developer portal** right above the details pane.
2. In the Developer portal, click on **APIS** and then the name of your API.
3. Click on the **Contacts\_Get** method
4. Click on the **Try it** button.
5. Leave the **Query parameters** and **Headers** set to their default values, scroll down, and click **Send**.
6. Scroll down to the **Response**. You’ll see it shows a status of **200 OK**, and the **Response content** of the API call.

Task 4: Publish new API Product

In this Task, you will create a new API Product within the API Management service, and publish it for use.

1. Within the Azure Portal, find and then click on the **API Management Service**.
2. On the **API Management Service** blade, under the **API MANAGEMENT** menu area, click **Products**. Then, click **+ Add** in the new blade.
3. In the **Add product** dialog, enter the following values:
   * Display name: **Free Trial**
   * ID: **free-trial (this should auto-populate)**
   * Description: **This is a free trial!**
   * State: **Published**

Then, click **Create**

1. As soon as the portal notifies you that the product has been created, in the **Products** page, click on the **Free Trial** product.
2. In the header bar of the **Free Trial** product, click **+ Add API**. Select the API you created.

Task 5: Register Developer Account

In this Task, you will register a new Developer account on the API Management service.

1. Navigate to the Azure Portal at [http://portal.azure.com](http://portal.azure.com/)
2. Within the Azure Portal, click on **Resource groups**, then the **LABAPIGATEWAY** resource group, then click on the **API Management service**.
3. On the **API Management service** blade, copy the **Developer portal URL**
4. Open a **New** browser window in **Private or Incognito Mode**, and navigate to the
5. Click on the **SIGN IN** link on the right side of the header navigation bar.
6. Click the **Sign up now** link.
7. On the **Sign up** page, enter the following values:
   * Email: **Enter your email address where you can receive email**
   * Password: **Enter a password**
   * First name: **Enter your first name**
   * Last name: **Enter your last name / surname**
   * Fill in the CAPTCHA accordingly.

Then, click **Sign Up** at the very bottom.

1. Open up your email account that you entered for the **Email Address** for the newly created login, and find the **Confirmation Email**.

**NOTE:** If using email services like **Gmail** or **Outlook**, you’ll want to open those within a **Private or Incognito Mode** browser tab so the login of the API Management Developer Portal for the login confirmation process doesn’t conflict with any other Azure logins.

1. Click the **Confirmation Link** in the email (if in a Private or Incognito mode browser), otherwise **Copy and Paste** the link into a **Private or Incognito mode** browser. *Be sure to use the same Private or Incognito browser the API Management Developer Portal is opened in.*
2. Once the account **Confirmation has completed** the browser will display the **Account Profile** page for the newly created Developer Account.

Task 6: Add Subscription to Developer Account

In this Task, you will add an API Product Subscription to the newly created Developer Account.

1. Within the **API Management service** blade in the Azure Portal, in the **API MANAGEMENT** menu area, click **Users**.
2. On the **Users** blade, select the user you just created.
3. On top of the new blade for your user, click **+ Add Subscription** button on the header menu.
4. On the **New Subscription** blade, give the subscription a meaningful name. Click in the **Product** field and select **free-trial** in the new dialog. Finally, click **Save**.
5. Once completed, the new User Account with have a **Subscription to the Starter product** within the API Management service.

Task 7: Retrieve Developer Subscription Keys

In this Task, you will retrieve the **Subscription Keys** for the Developer Account that are necessary when making API calls against the service.

1. Open the **Developer portal** in a **Private or Incognito mode** browser window.

NOTE: Remember, you can find the link to the Developer portal in the **Overview** screen of the **API Management** blade in the Azure Portal.

1. Click the **SIGN IN** link on the right of the top navigation.
2. **Sign in** with the **Developer Account**.
3. Once logged in, click on the **Username** in the upper-right, and click on **PROFILE**.
4. Under the **Subscription details**, click on the **Show** link for the **Primary Key**.
5. Copy the **Primary Subscription Key** for use in a later task.

Task 8: Retrieve Managed API Swagger Definition File

In this Task, you will download the Swagger service definition file for the Managed API within the API Management service.

1. Within the **Developer portal**, click on **APIS** in the upper navigation.
2. Click on the name of your API.
3. On the right side of the page, click on the **API definition** button, and click on the **OpenAPI** option.
4. Save the **OpenAPI Definition** file for use in the next exercise by right-clicking in the resulting tab and saving it as a JSON file.

Task 9: Generate Client SDK

In this Task, you will use Swagger and Visual Studio to generate a Client SDK for your Manage API that will be used within a .NET Console Application.

1. Open a new instance of **Visual Studio**, click on the **File** menu, then **New**, and click **Project…**
2. In the **New Project** dialog, navigate to the **Windows Classic Desktop** project template section and select the **Console App** template.
3. Type in a **Name** for the Console Application project, then click **OK**.
4. In the **Solution Explorer** window, right-click on the project, then click on **Add**, then **REST API Client…**
5. On the **Add REST API Client** dialog, select the **existing Swagger metadata file** option and click the **Browse…** button.
6. Locate and select the **Swagger API Definition File** that was downloaded in the previous exercise.
7. Click the **OK** button.
8. Once the **Client SDK** code has been generated, it will be saved into a subfolder within the Project with the same name as the **Managed** **API.** You can navigate to this folder to see the full Client SDK source code.
9. Open the **Program.cs** source code file.
10. Paste in the following line of code to the **void Main** method to declare the **Subscription Key** to use with the API Client. Be sure to replace **{subscription-key}** with the Primary Key copied from a previous exercise.

var subscriptionKey = "{subscription-key}";

1. Next, add the following line of code to the **void Main** method to instantiate a new instance of the **API Client**. Use the namespace you specified before when adding the REST API Client. The class name used will be the same name as the Managed API amd be capitalized:

var apiClient = new LabWebApi(

new Microsoft.Rest.TokenCredentials(subscriptionKey)

);

1. Paste in the following line of code to create a new **Contact** object that will be used to insert a new record using the API.
2. var newContact = new Models.Contact
3. {
4. PartitionKey = "John",
5. RowKey = "Johnson",
6. City = "Milwaukee",
7. StateProv = "WI",
8. PrimaryPhone = "555-1234"

};

1. Paste in the following line of code that uses the **Put** action method on the **Contacts** **API** to insert a new **Contact** record.

apiClient.Contacts.Put(subscriptionKey, contact: newContact);

1. Press **Ctrl-F5** to debug the console app and ensure it executes without error.
2. Delete the previous 2 lines of code, leaving the **apiClient** and **subscriptionKey** variable declaration lines.
3. Paste in the following code that retrieves **All Contacts** using the **Get** action method on the **ContactsAPI**, then writes them out to the Console.

var allContacts = apiClient.Contacts.Get(subscriptionKey);

foreach (var contact in allContacts)

{

Console.WriteLine(

string.Format("{0}, {1} - {2}, {3}",

contact.RowKey, // first name

contact.PartitionKey, // last name

contact.City,

contact.StateProv

)

);

}

1. Press **Ctrl-F5** to debug the console app and see the output of all contacts in the Console.

![](Exercise3images/media/image62.png)

Summary

In this exercise, you created an API Management instance, imported the API created in the previous exercise, made a test call to the API using the Developer Portal, published a new API product, registered a new, non-admin developer account, subscribed to the new API product with the new developer account and eventually created a new client SDK that used the developer account’s subscription key to make a call to the API.