# Microsoft Cloud Workshop

## Media AI

Hands-on lab step-by-step

March 2018

# Contents

# Media AI hands-on lab step-by-step

## Abstract and learning objectives

In this workshop, students will learn how to build, setup, and configure a Web Application that performs media streaming using Azure Services; including the Video Indexer API. Students will also learn how to implement video processing using Logic Apps, Azure Functions, and Video Indexer API to encode and transcribe videos.

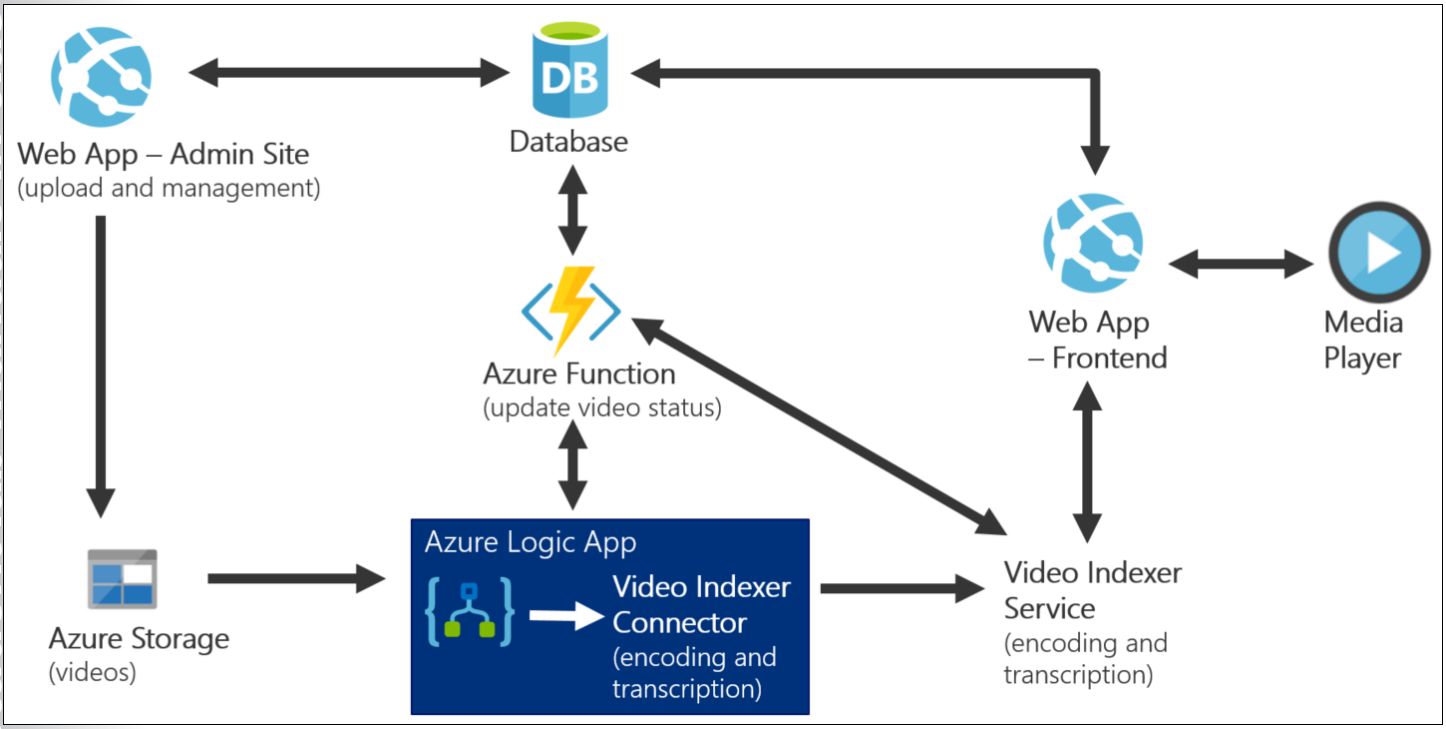
Students will be better able to build media applications including:

- Setup Video Indexer API
- Upload videos to Blob Storage to be encoded with Azure Video Indexer
- Integrate Video Indexer through Logic Apps and Azure Functions

## Overview

Contoso has asked you to build a media streaming service so they can deliver their on-demand video training courses to their customers. For this solution, Contoso wants to use PaaS and Serverless services within the Microsoft Azure platform to help minimize development time and increase ease of maintenance. They would also like you to integrate automatic transcript generation and caption translation within the video encoding process so overall video production cost can be reduced, as well as improving the video player experience for their customers across the globe.

# Solution architecture




# Requirements

- Microsoft Azure subscription
- Local machine or Azure LABVM virtual machine configured with:
    - Visual Studio 2017 Community Edition or later

# Before the hands-on lab

Duration: 30 minutes

In this lab, you will create a developer environment and download the required files for this course if you do not already have one that meets the requirements.

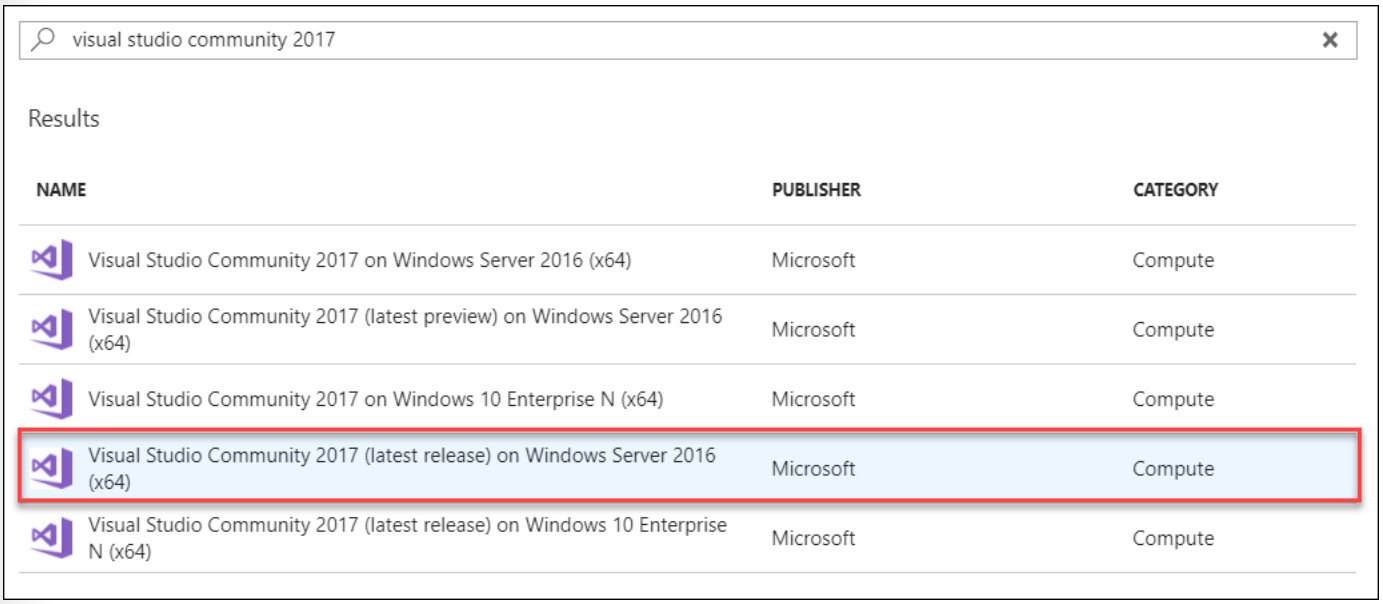
## Prerequisites

1. Microsoft Azure subscription http://azure.microsoft.com/en-us/pricing/free-trial/
2. Client computer with Windows 7 or later with Visual Studio 2017

## Task 1: Configure a development environment

If you do not have a machine setup with Visual Studio 2017 Community complete this task.

1. Create a virtual machine in Azure using the **Visual Studio Community 2017 (latest release) on Windows Server 2016 (x64)** image. This is important as you need to have Visual Studio 2017 version 15.4 or later to complete this lab.



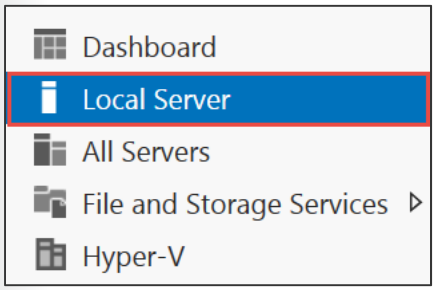

It is **highly** recommended to use a DS2 or D2 instance size for this VM.

## Task 2: Disable IE enhanced security

Note: Sometimes this image has IE ESC disabled, and sometimes it does not.

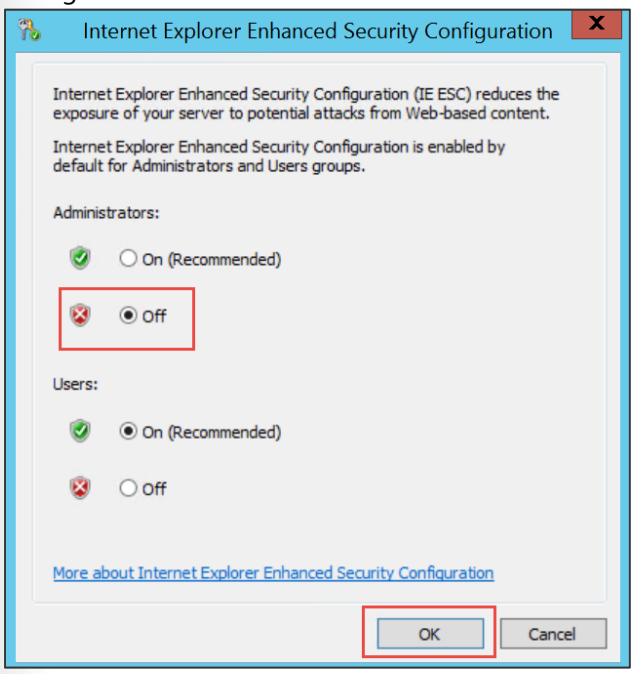
1. On the new VM, you just created click the Server Manager icon

2. Click Local Server



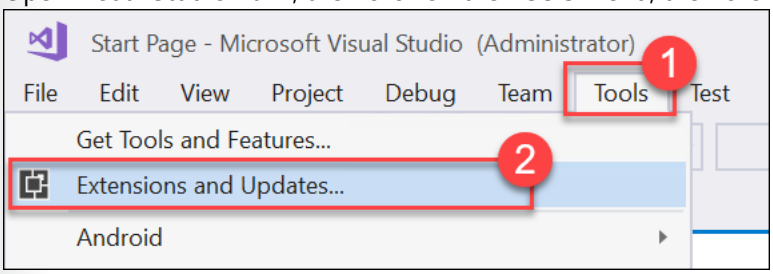

3. On the right side of the pane, click **On** by IE Enhanced Security Configuration


4. Change to **Off** for Administrators and click **OK**

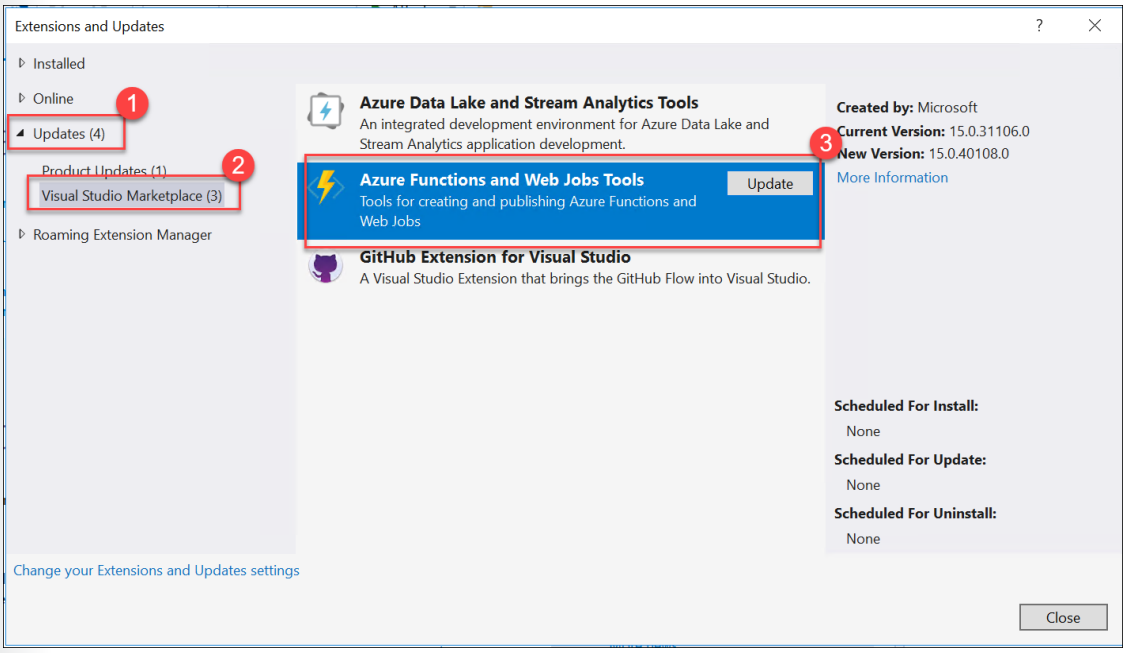



## Task 3: Update Visual Studio Tools for Azure Functions

1. Open Visual Studio 2017, then click on the **Tools** menu, then click on **Extensions and Updates...**

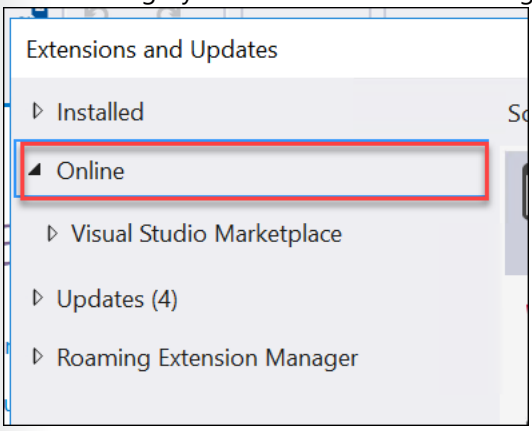


2. On the Extensions and Updates dialog, click on **Updates** then **Visual Studio Marketplace** on the left side of the dialog, then click on **Update** for the **Azure Functions and Web Jobs Tools** extension to update to the latest version.

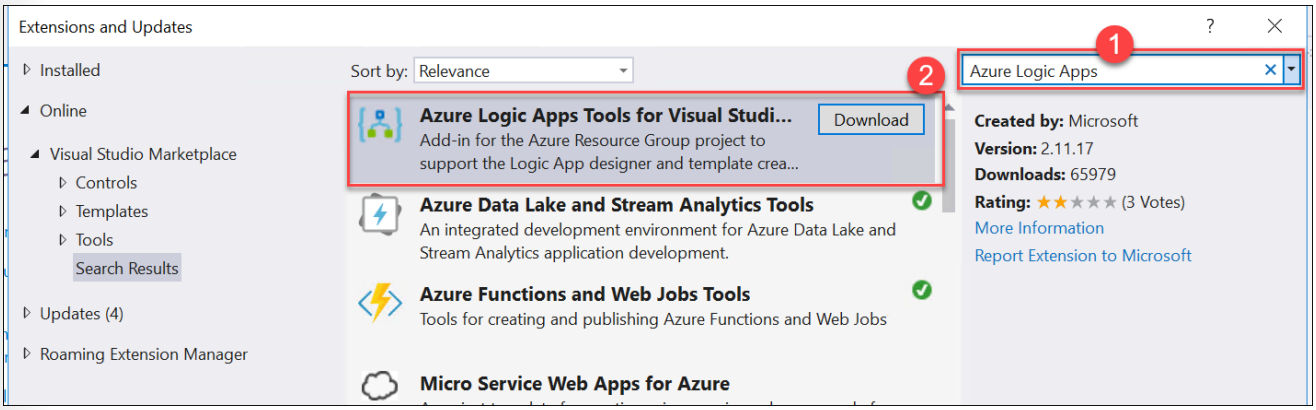



## Task 4: Install Visual Studio Tools for Logic Apps

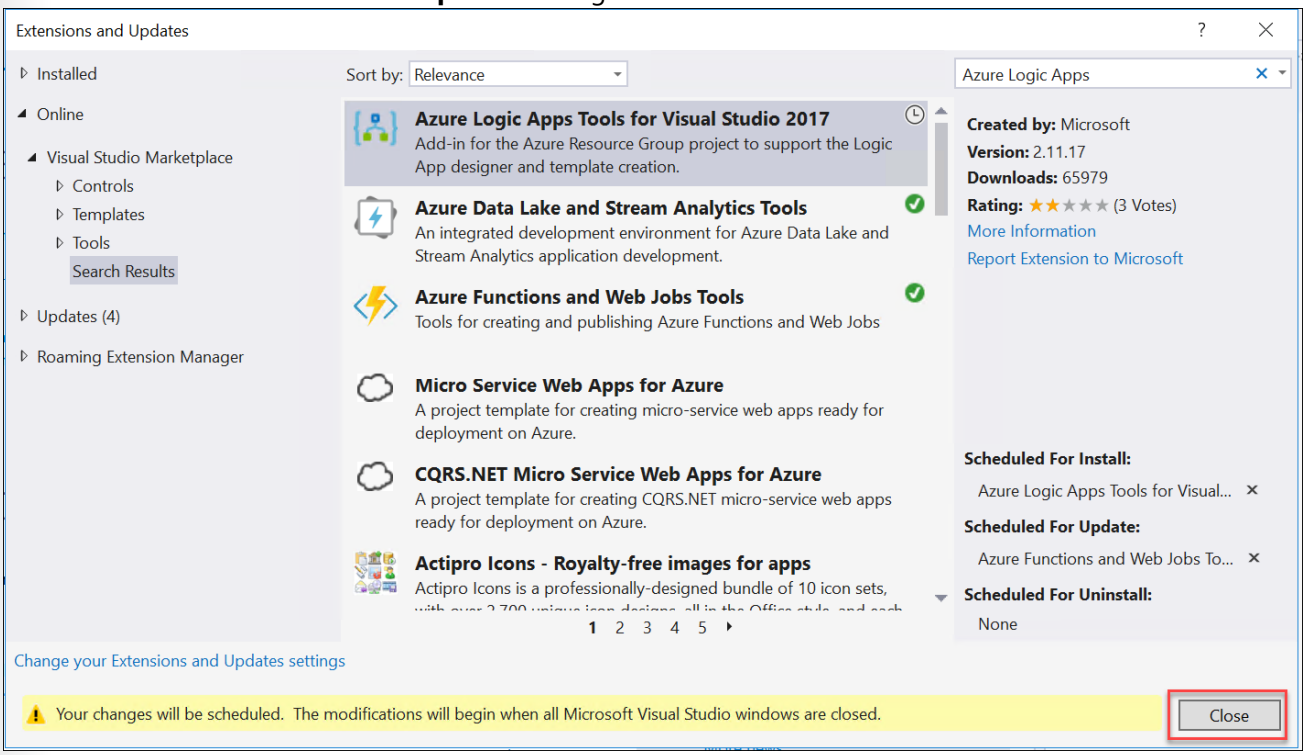
1. Within the **Extensions and Updates** dialog within Visual Studio 2017 sill open from the previous task, click on the **Online** category on the left side of the dialog



2. In the **Search** box in the upper right of the dialog, type in **Azure Logic Apps** to search for the **Azure Logic Apps Tools for Visual Studio** extension. Then click **Download** on the Extension to install it.

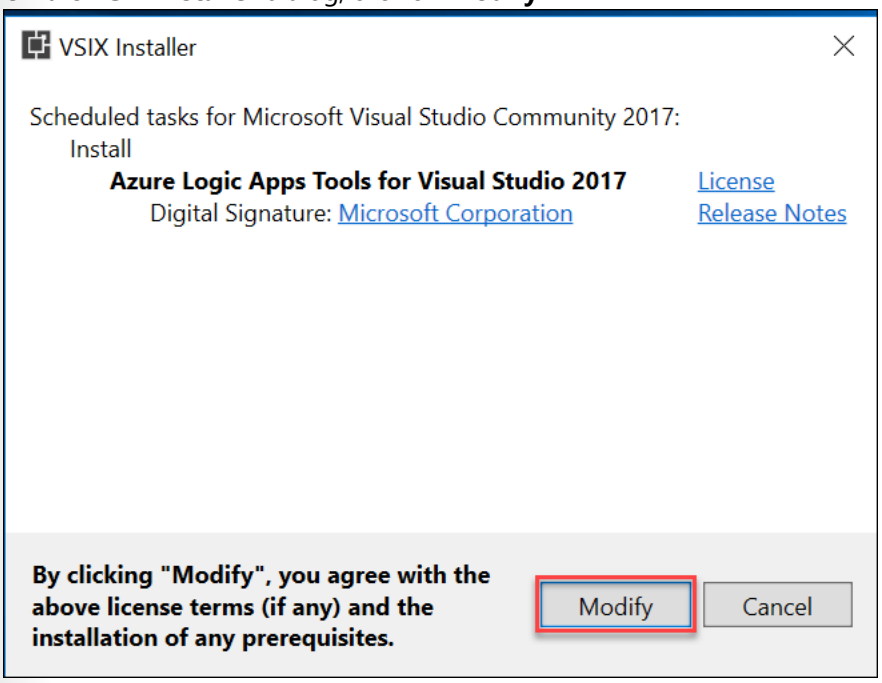



3. Click **Close** on the **Extensions and Updated** dialog

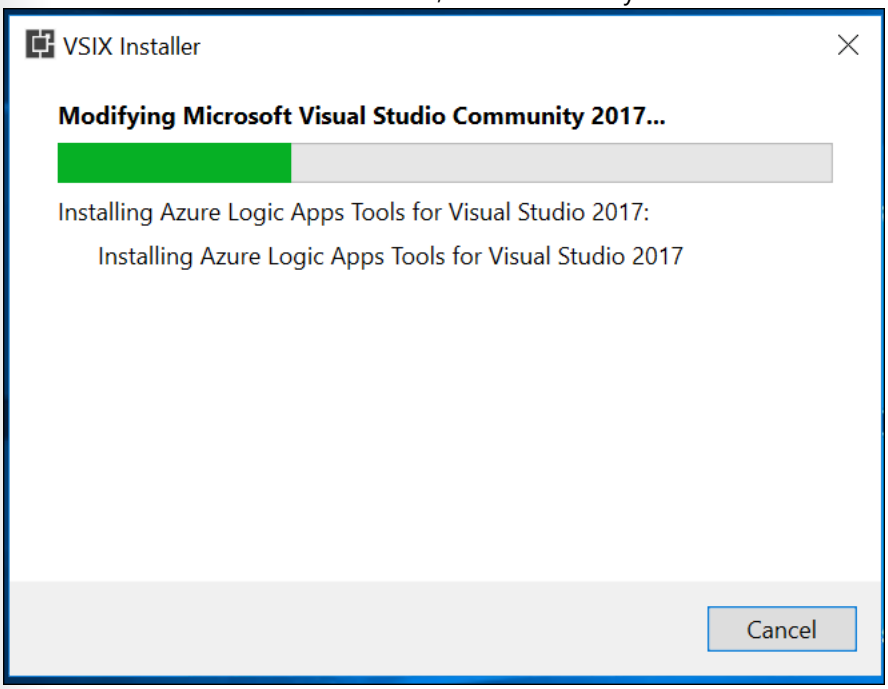



4. Close Visual Studio, then the pending installation of the Azure Logic Apps Tools for Visual Studio will automatically launch

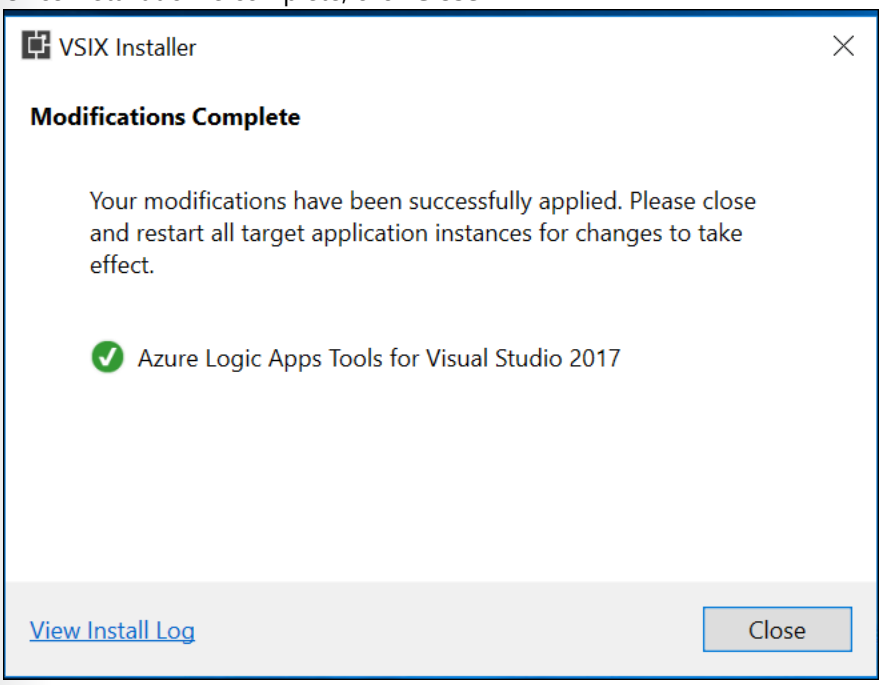
5. On the **VSIX Installer** dialog, click on **Modify**




6. Wait for the extension to be installed, this should only take about 1 minute



7. Once installation is complete, click **Close**

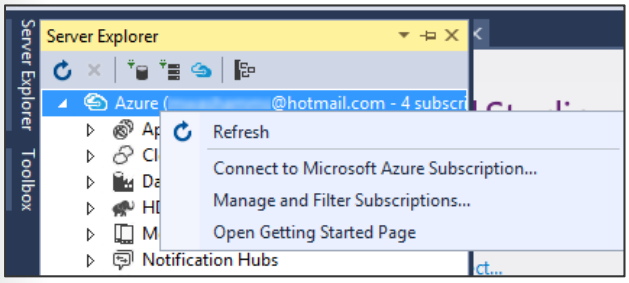



8. Restart Visual Studio now that the extension has been installed

## Task 5: Validate connectivity to Azure

1. From within the virtual machine, Launch Visual Studio 2017 and validate that you can login with your Microsoft Account when prompted

2. Validate connectivity to your Azure subscription. Launch Visual Studio, open Server Explorer from the View menu, and ensure that you can connect to your Azure subscription.




## Task 6: Download the exercise files

1. Download the exercise files for the training (from within the virtual machine)
    a. Create a new folder on your computer named **C:\Hackathon**
    b. Download the support files (.zip format), https://cloudworkshop.blob.core.windows.net/media-services-and-cdn/Media-Services-Student-Files.zip to the new folder
    c. Extract the contents to the same folder

# Exercise 1: Signup for Video Indexer API Service

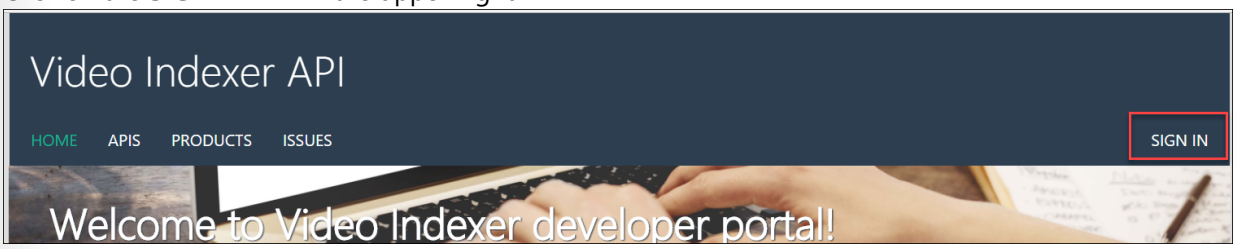
Duration: 15 minutes

In this exercise, you will setup the Video Indexer API within Microsoft Azure.

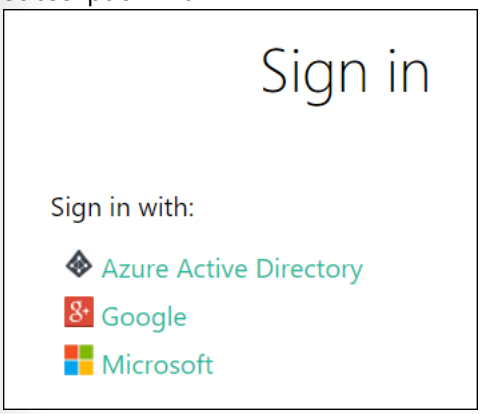
## Task 1: Signup for Video Indexer

1. Open a new browser window / tab and navigate to https://videobreakdown.portal.azure-api.net/

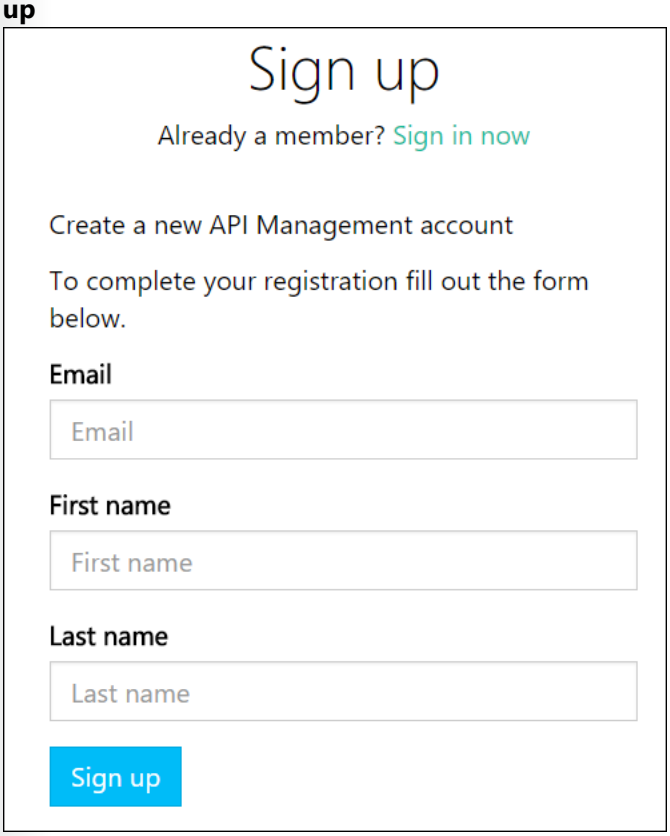
2. Click on the **SIGN IN** link in the upper-right



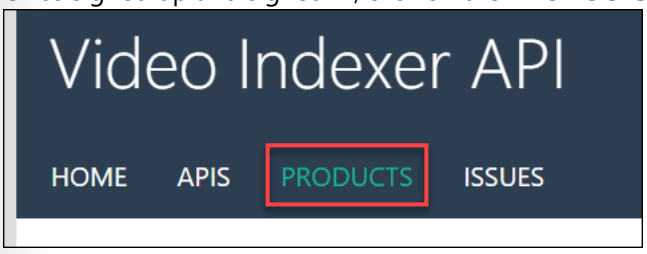

3. Click to sign in with your **Microsoft** or **Azure AD** credentials. Use the same credentials you login to your Azure Subscription with



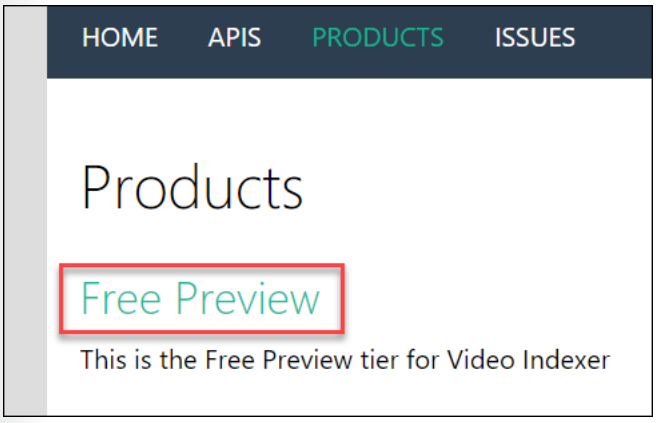
4. The first time you Sign in, you will be prompted to create a new account. Fill in the required fields, then click **Sign up**



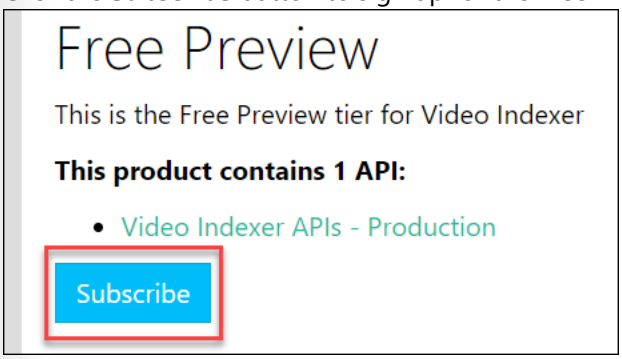

5. Once signed up and signed in, click on the **PRODUCTS** menu at the top



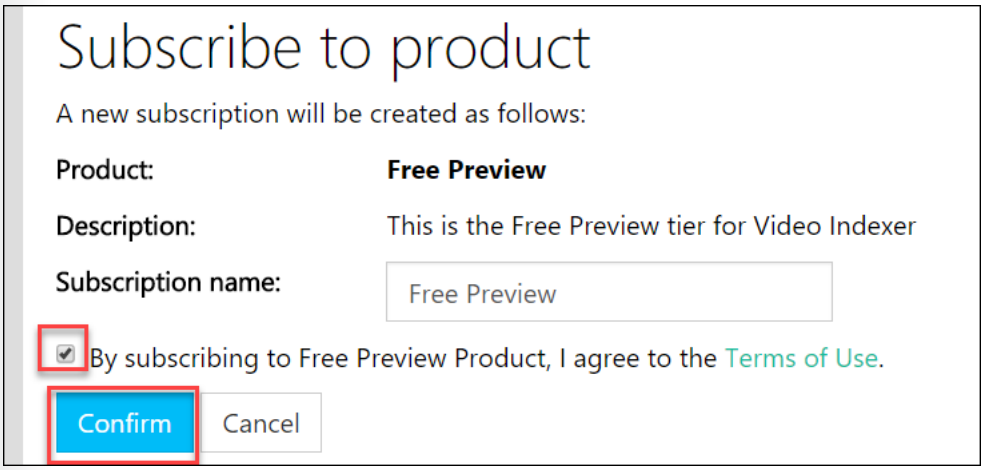

6. Click on the **Free Preview** Product link



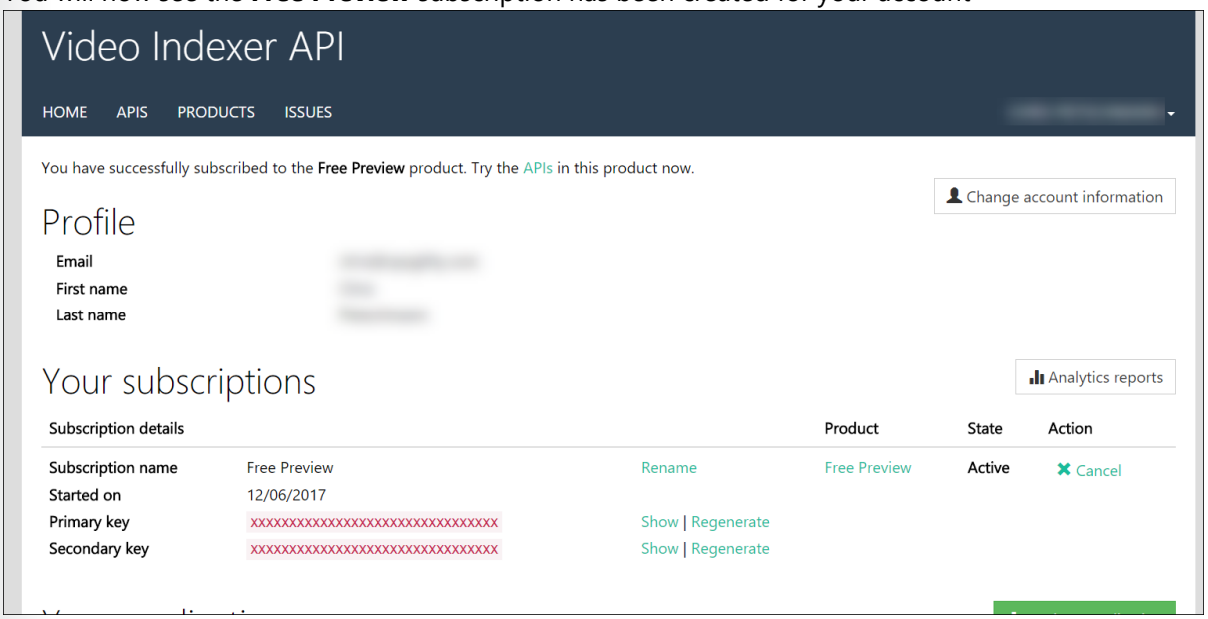
7. Click the **Subscribe** button to sign up for the Free Preview




8. Check the "**I agree to the Terms of Use.**" Box, then click the **Confirm** button to finish signing up for the Free Preview




9. You will now see the **Free Preview** subscription has been created for your account

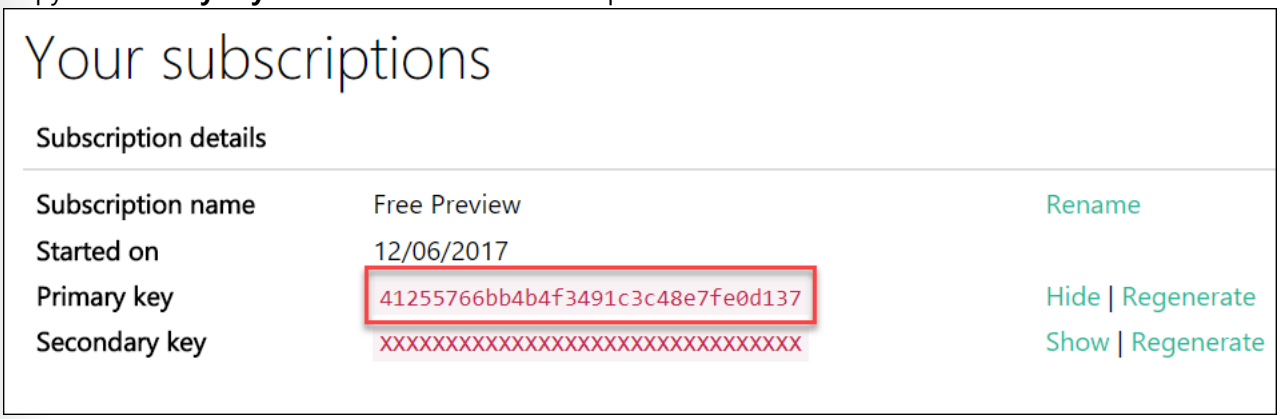


## Task 2: Copy Video Indexer API Key

1. While on the **Profile** page of the **Video Indexer API**, locate the **Subscription details** for the **Free Preview** subscription

2. Click the **Show** link next to the **Primary key** to reveal the API key


3. Copy the **Primary key** for the **Free Preview** subscription and save this for use later




# Exercise 2: Setup video import workflow

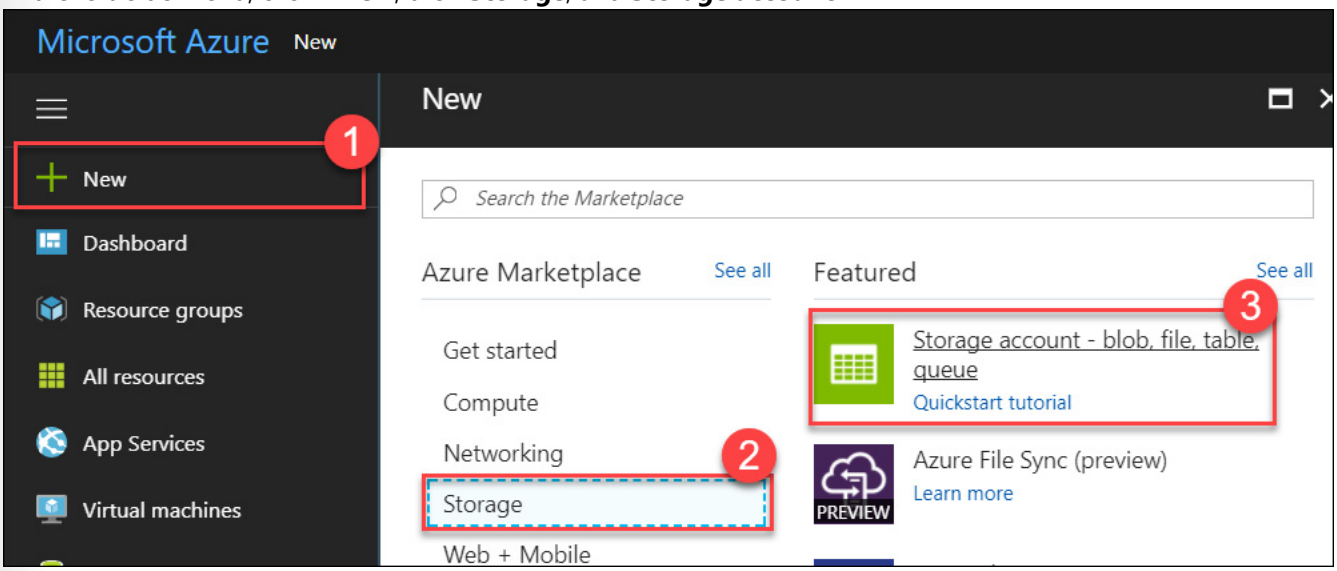
Duration: 20 minutes

In this exercise, you will set the import workflow for uploading and importing videos using the Video Indexer API.

## Task 1: Create Storage Account for video files

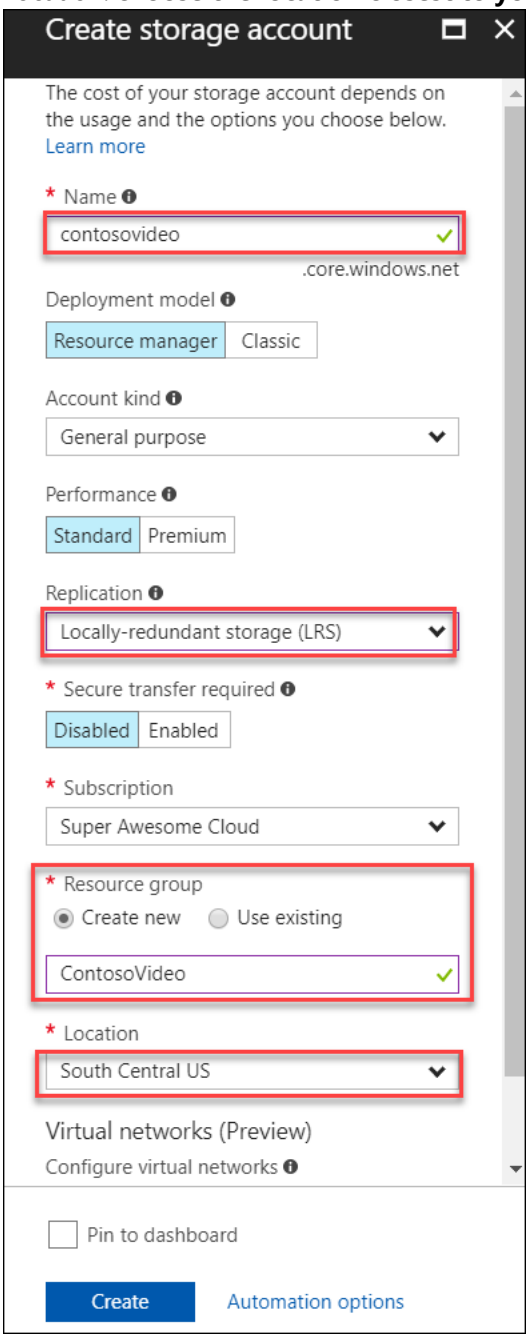
1. Open a browser window and login to the Azure Portal http://portal.azure.com

2. In the left-side menu, click **+New**, then **Storage**, and **Storage account**



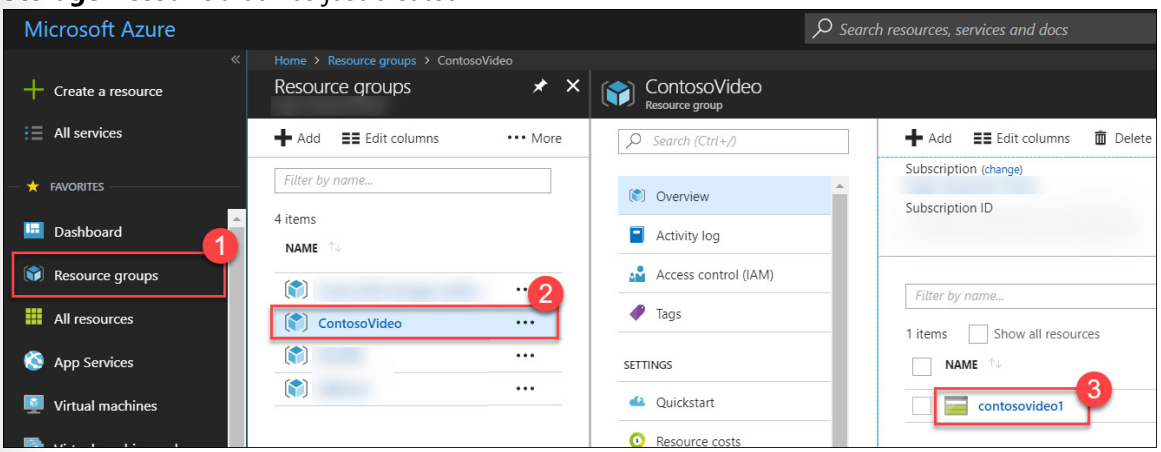

3. On the **Create storage account** blade, enter the following values:
   a. Name: **enter a unique name**
   b. Replication: **Locally-redundant storage (LRS)**
   c. Resource Group: **ContosoVideo**

d. Location: **choose the location closest to you**

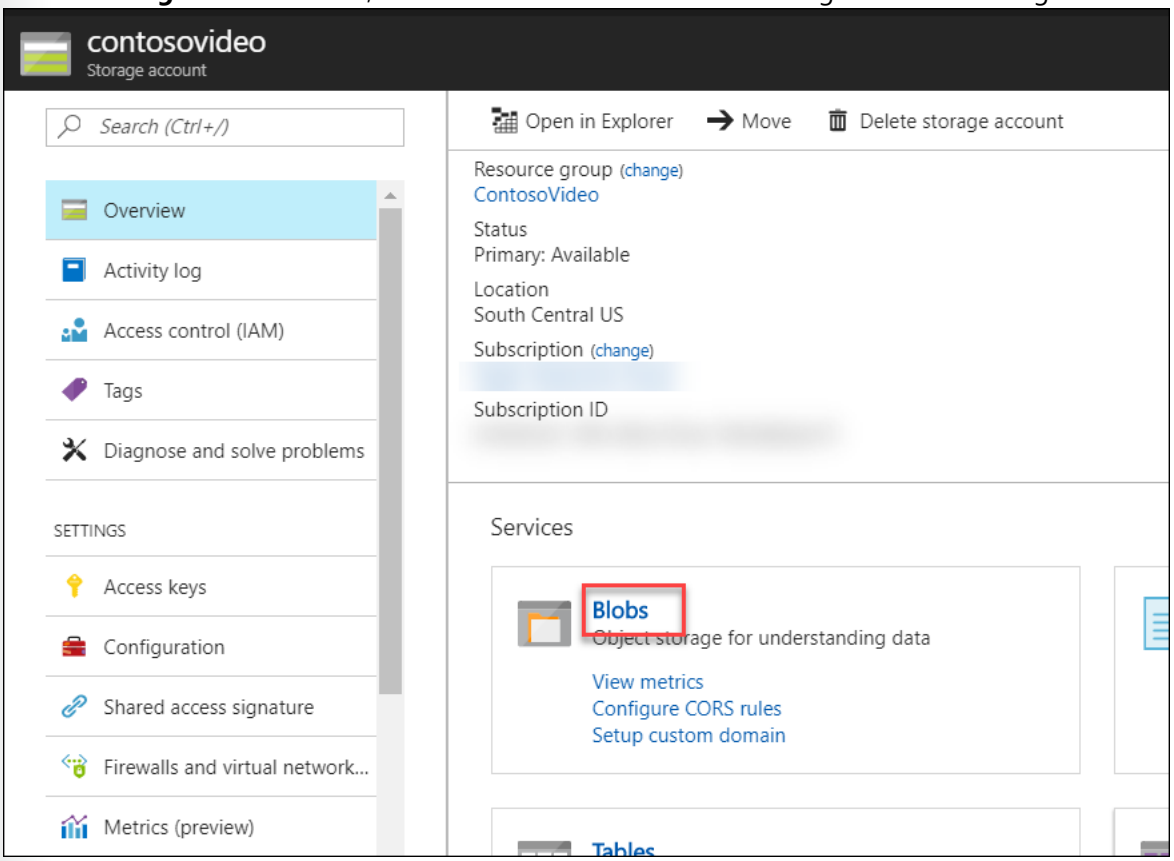



4. Click **Create**

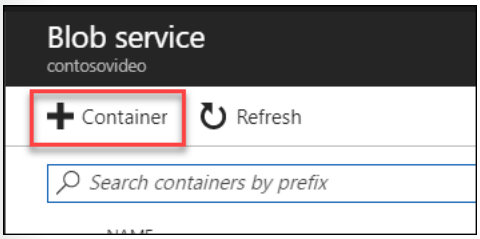
5. In the left-side menu, click **Resource groups**, then click on the **ContosoVideo** Resource group, then click the **Storage Account** that was just created



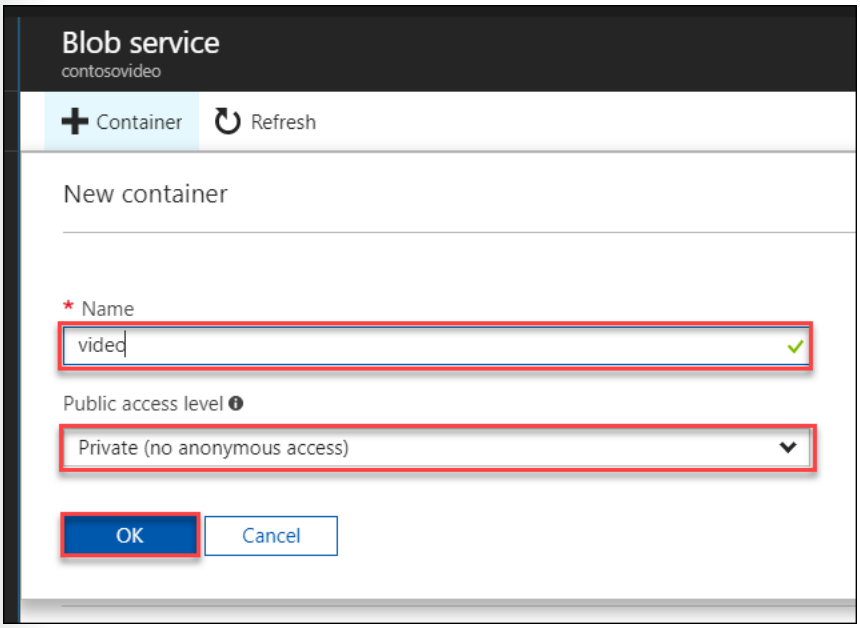

6. On the **Storage account** blade, click the **Blobs** link under Services to get started creating a new Blob Container



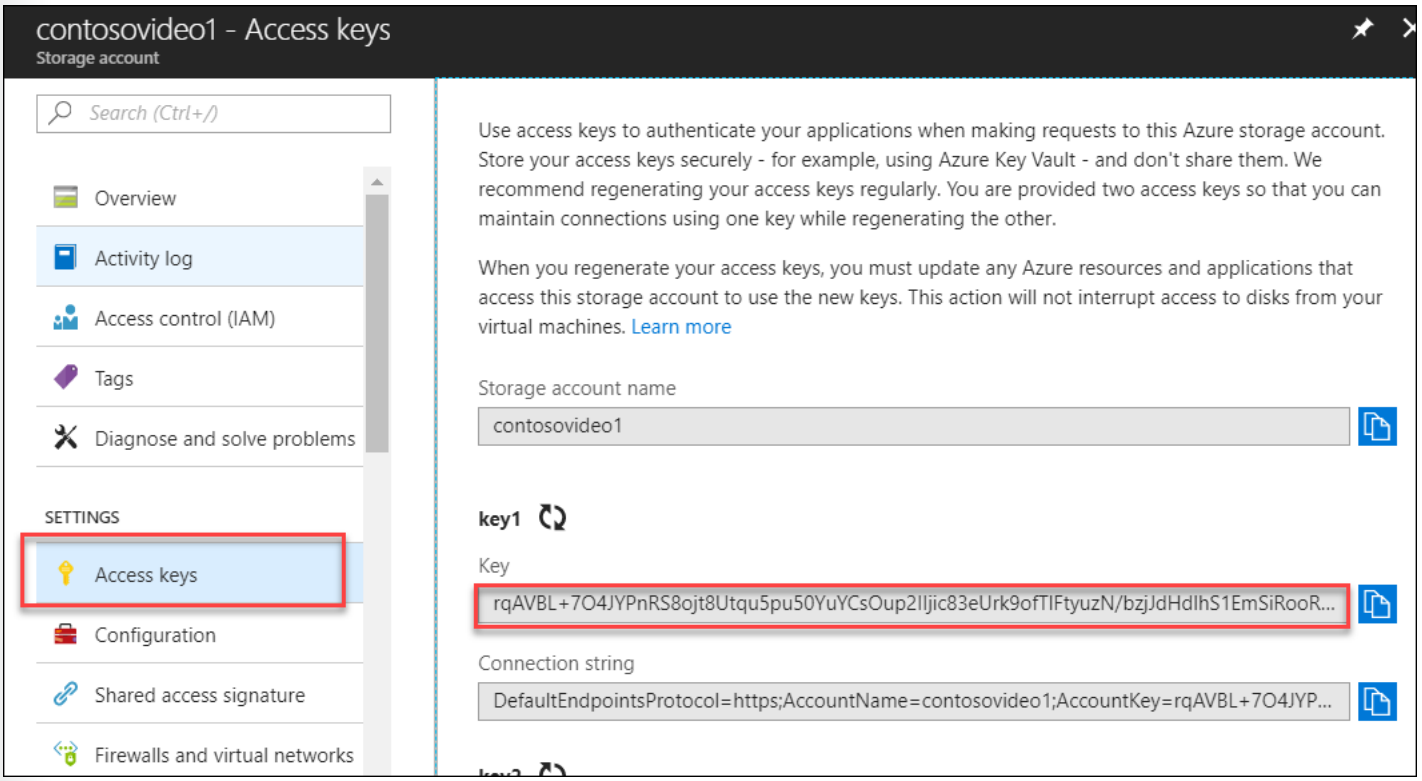

7. Click the **+Container** button



8. Enter **video** for the name of the new Container, leave the Public access level set to **Private**, then click **OK**



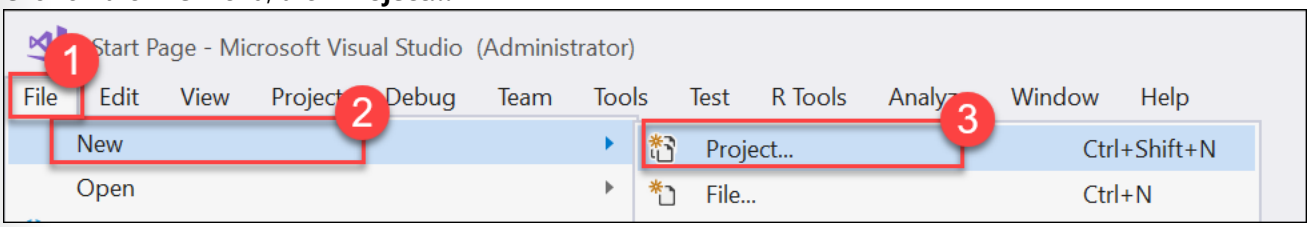

9. In the left pane of the Storage Account blade, click on **Access keys**, then copy the "primary" **key1**. Save this for reference later.



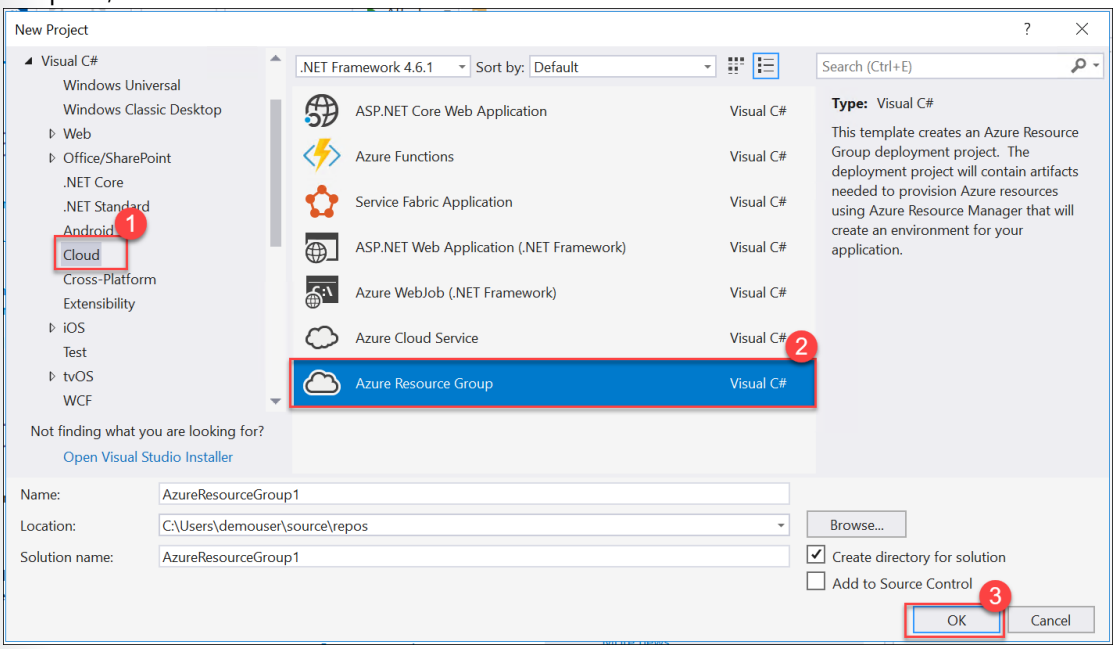

## Task 2: Create Azure Logic App to process videos

1. Open **Visual Studio 2017**

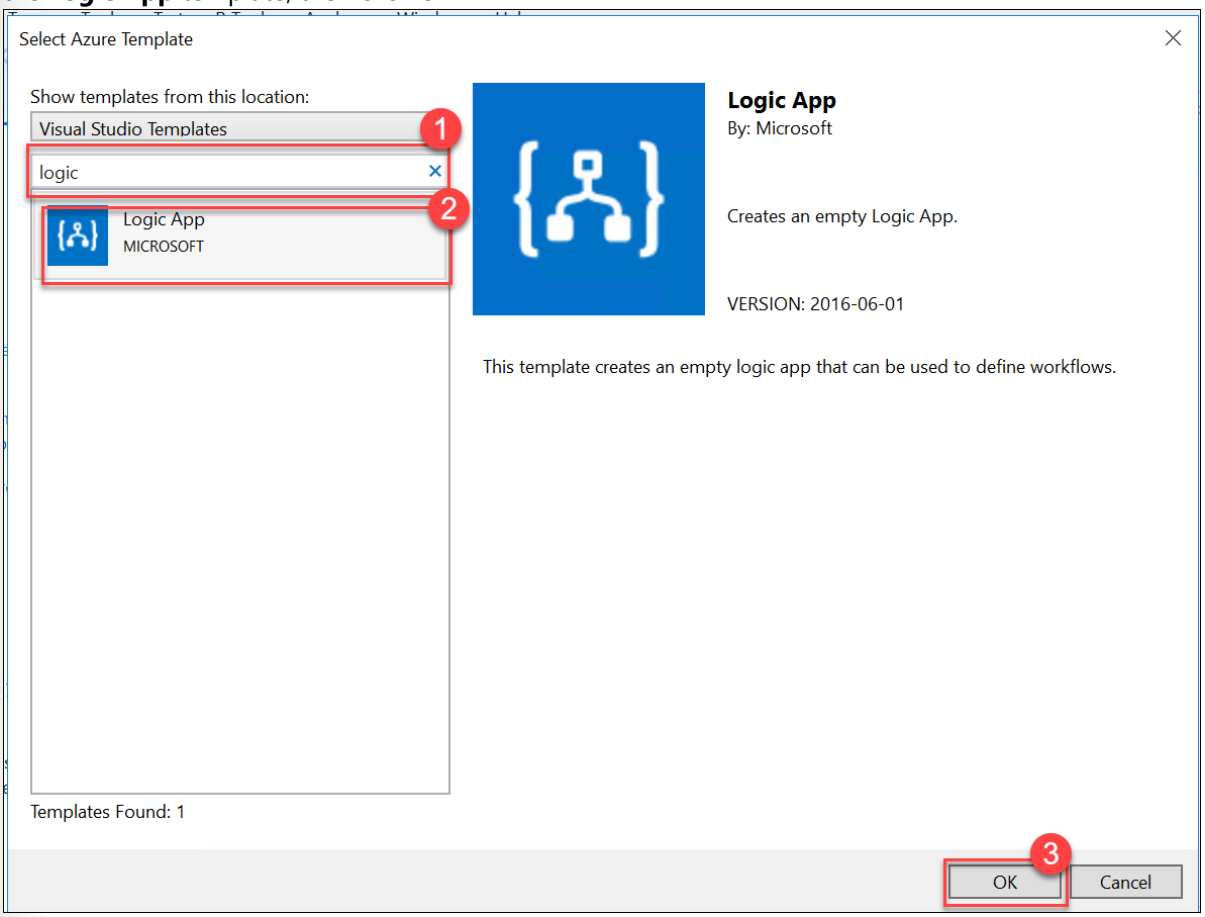
2. Click on the **File** menu, then **Project...**



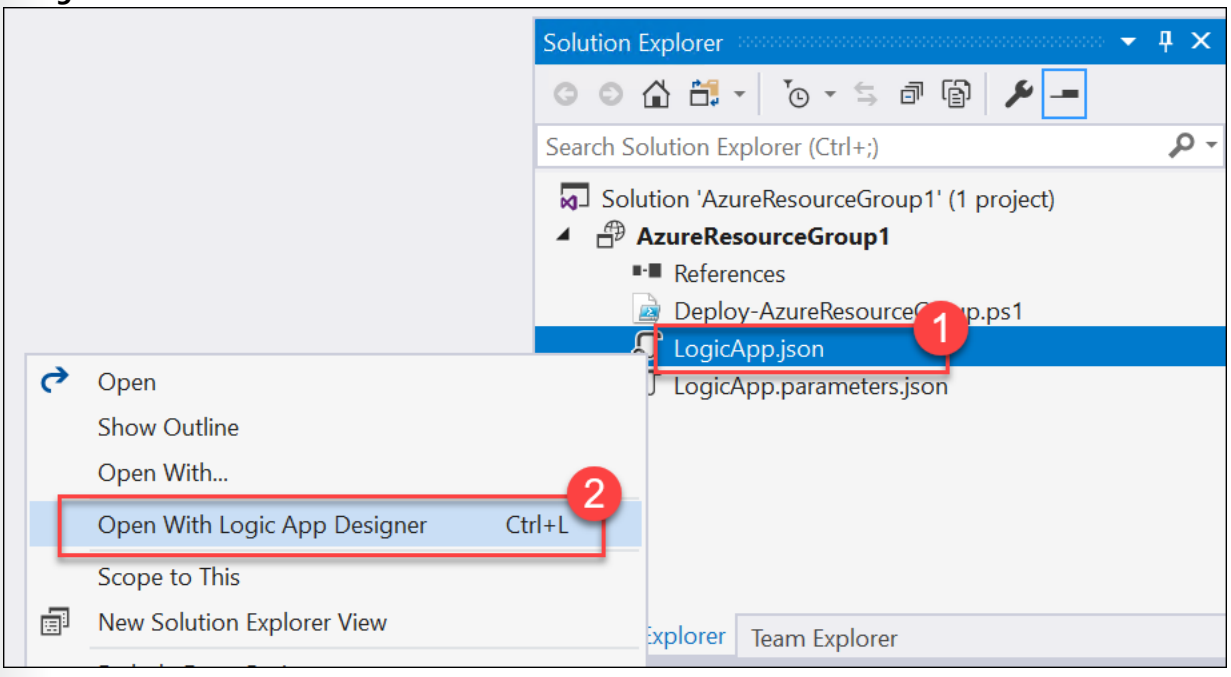

3. On the **New Project** dialog, select the **Cloud** category on the left, then select the **Azure Resource Group** project template, then click **OK**



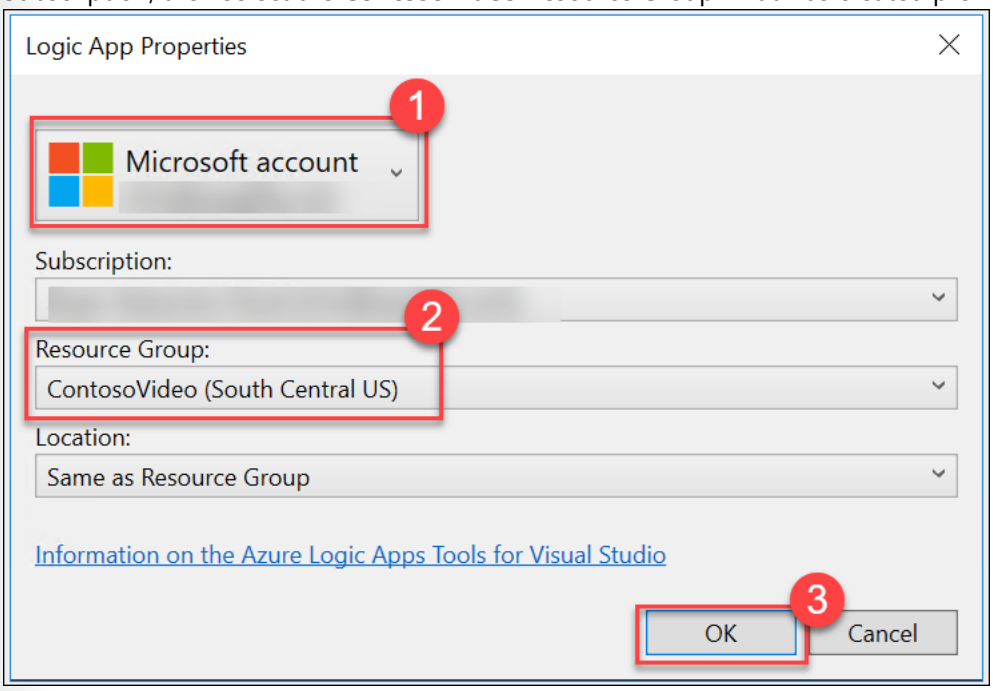
4. In the **Select Azure Template** dialog, type **Logic** into the Search box to filter the available templates, then select the **Logic App** template, then click **OK**



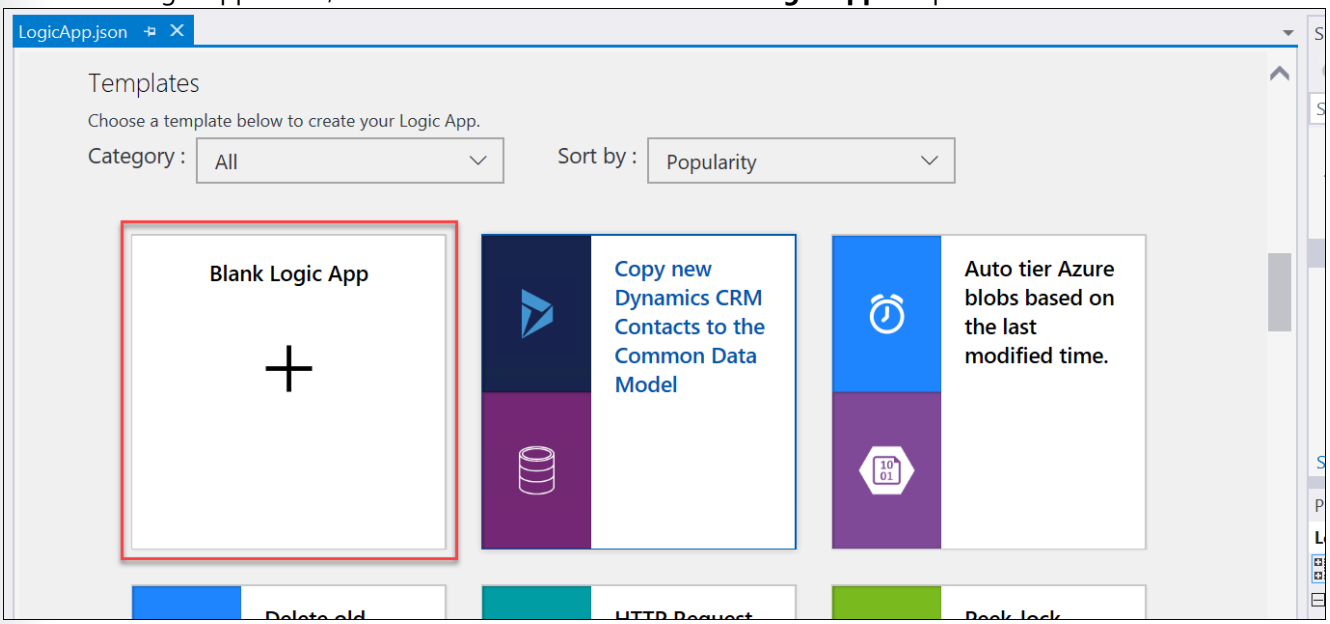

5. Within the **Solution Explorer** pane, right-click on the **LogicApp.json** file, then click on **Open With Logic App Designer**



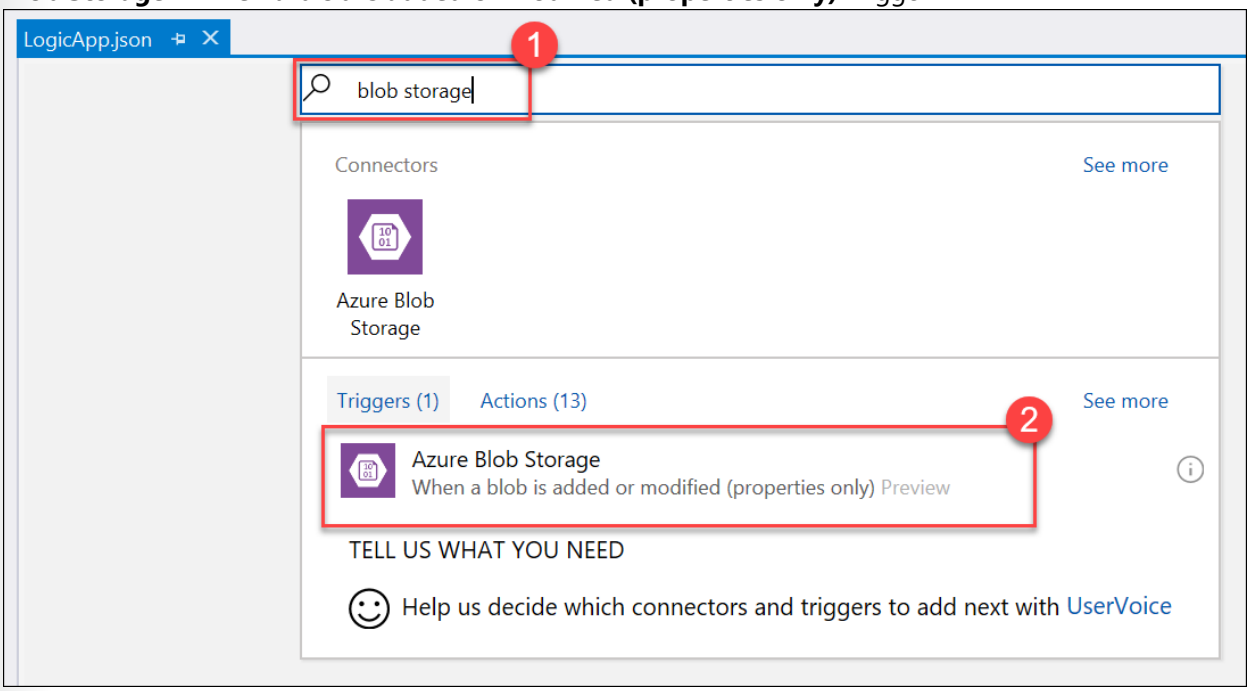
6. On the **Logic App Properties** dialog, sign into your Microsoft or Organization Account associated with your Azure Subscription, then select the **ContosoVideo** Resource Group what was created previously, then click **OK**



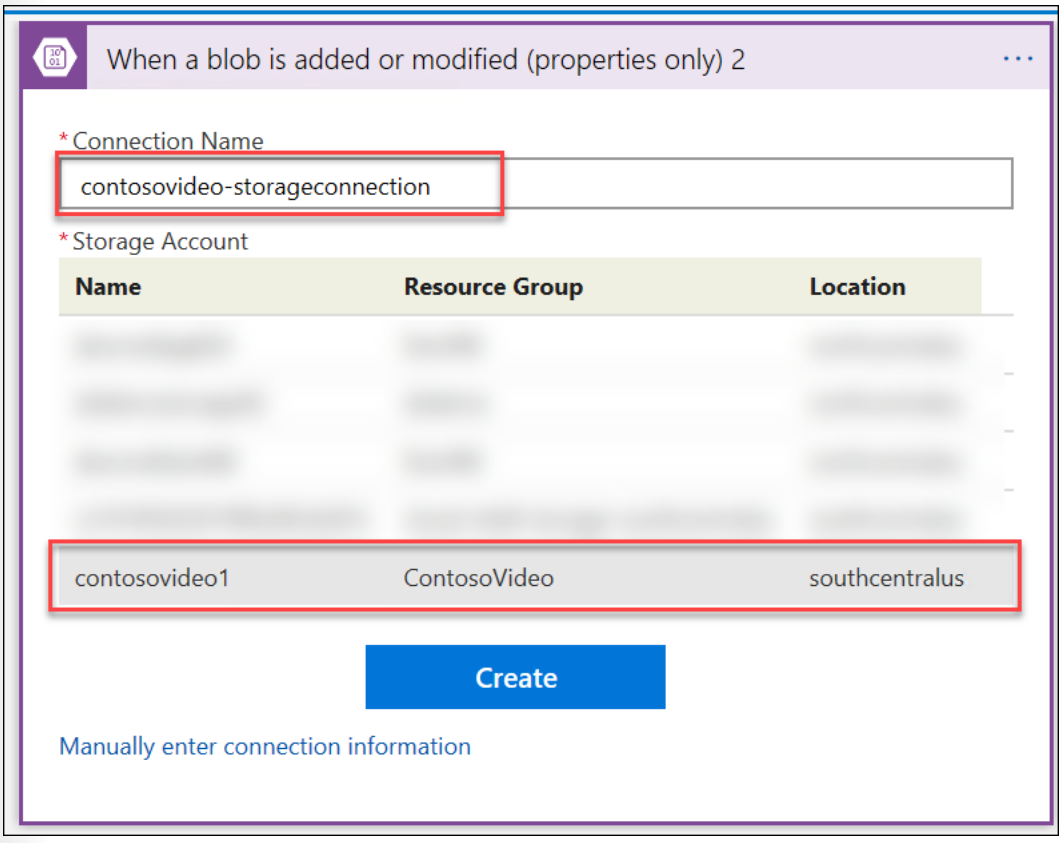

7. Within the Logic App Editor, scroll down and click on the **Blank Logic App** template



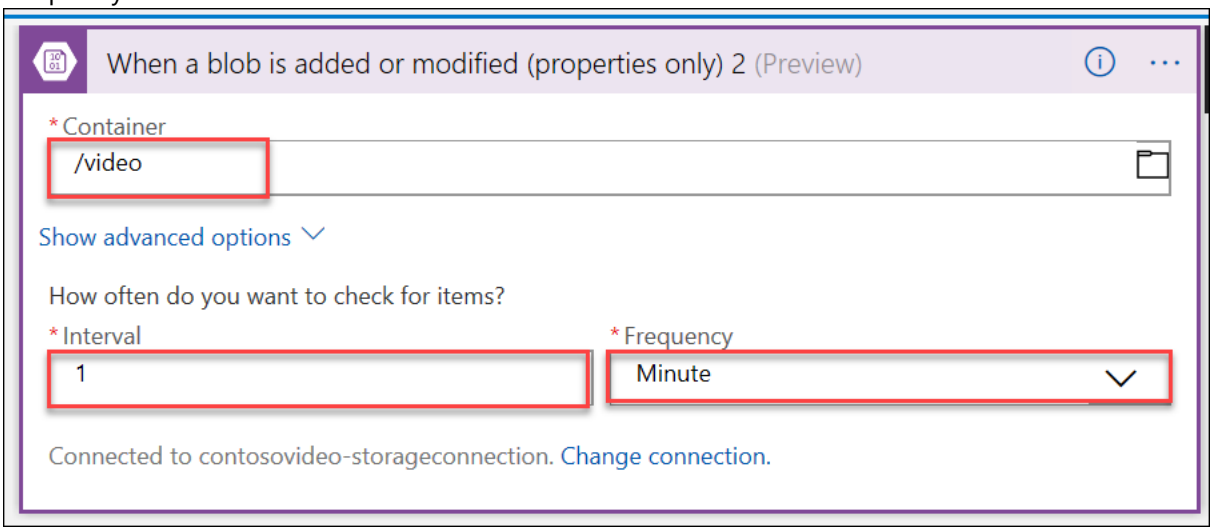
8. In the **Logic App Designer**, start building out the Logic App Workflow by searching for and adding the **Azure Blob Storage – When a blob is added or modified (properties only)** Trigger



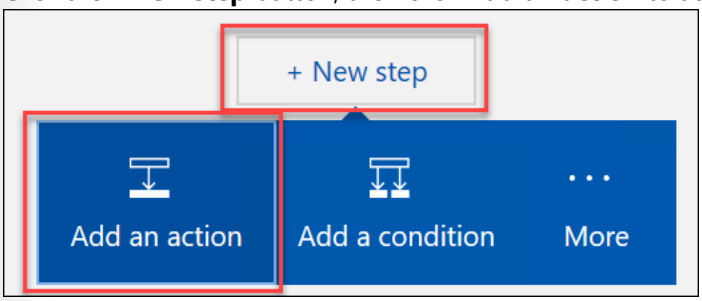

9. Enter a name in the **Connection Name** field, then click on the **Storage Account** that was previously created, then click **Create**



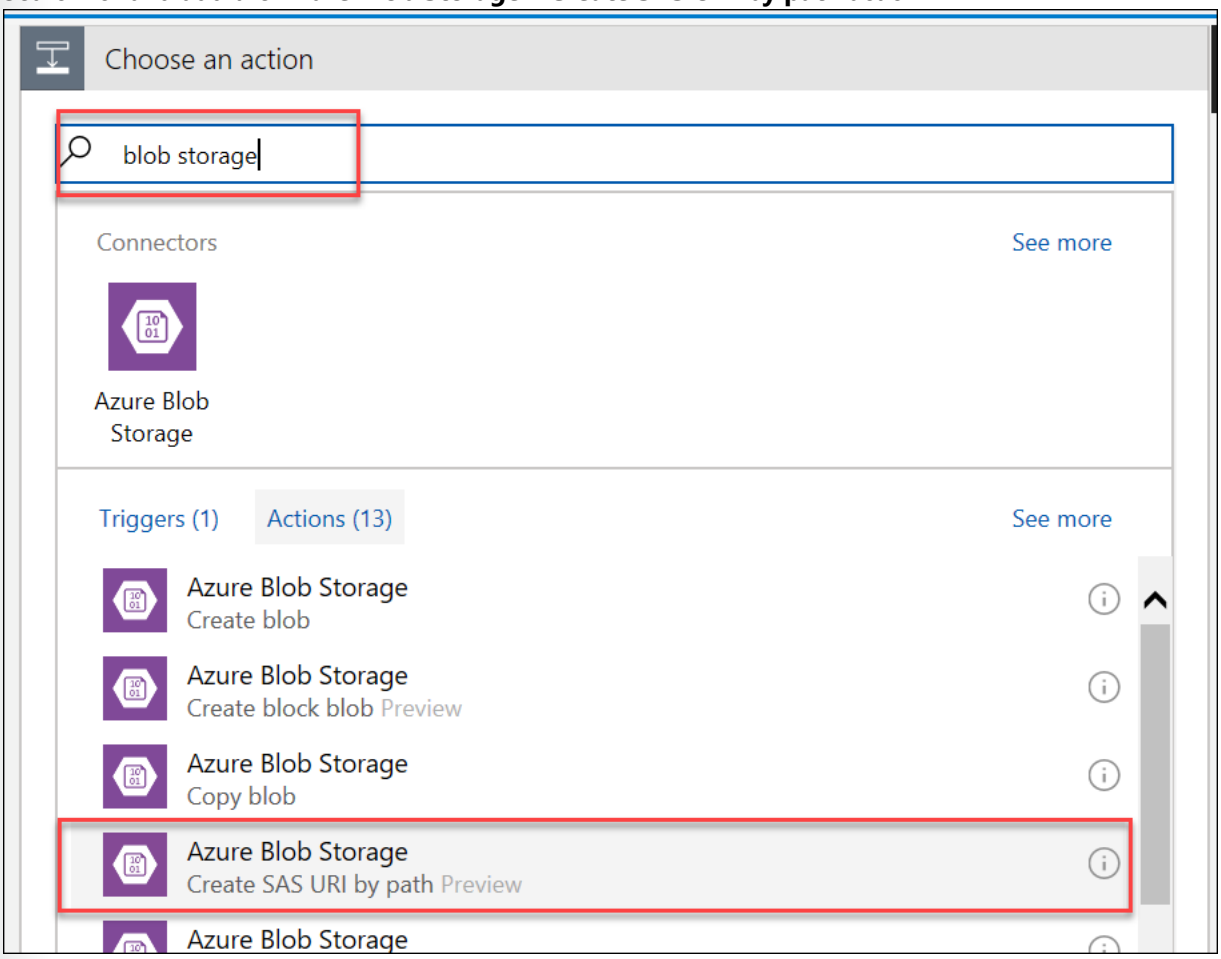
10. On the **Blob Storage Trigger**, enter the following values:
    a. Container: **video**
    b. Interval: **1**
    c. Frequency: **Minute**



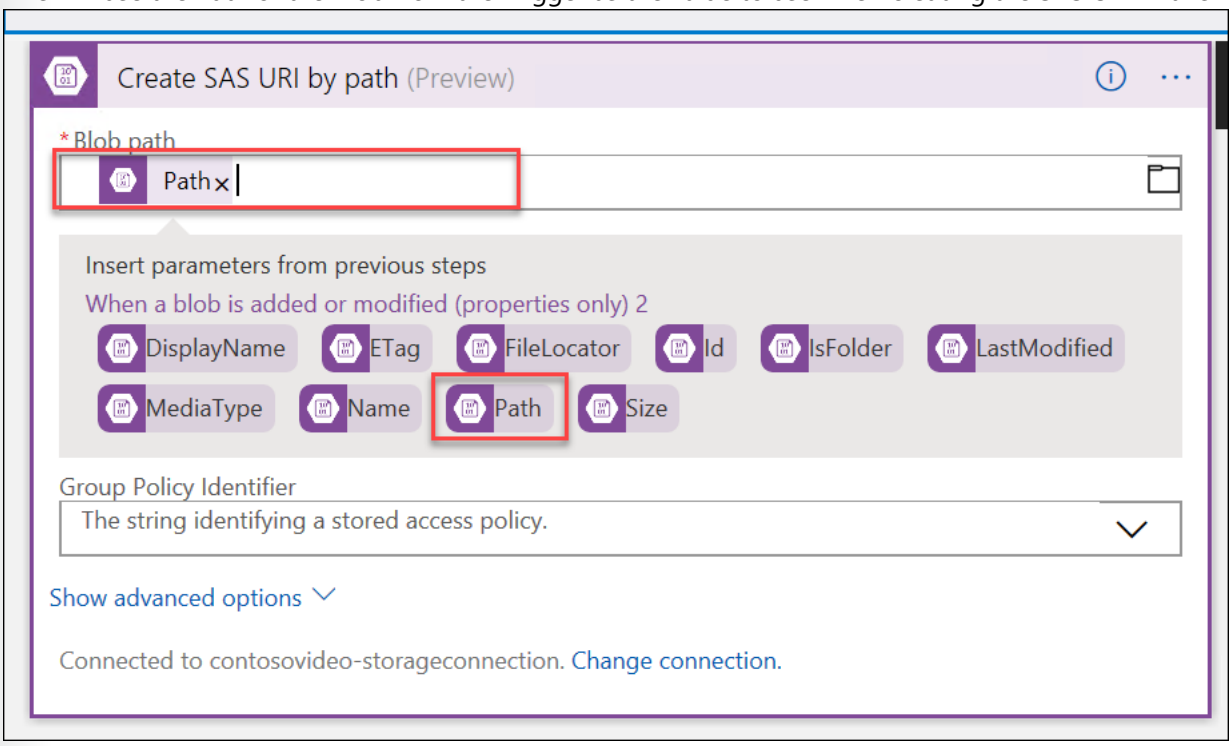

11. Click the **+New step** button, then click **Add an action** to add another action to the workflow



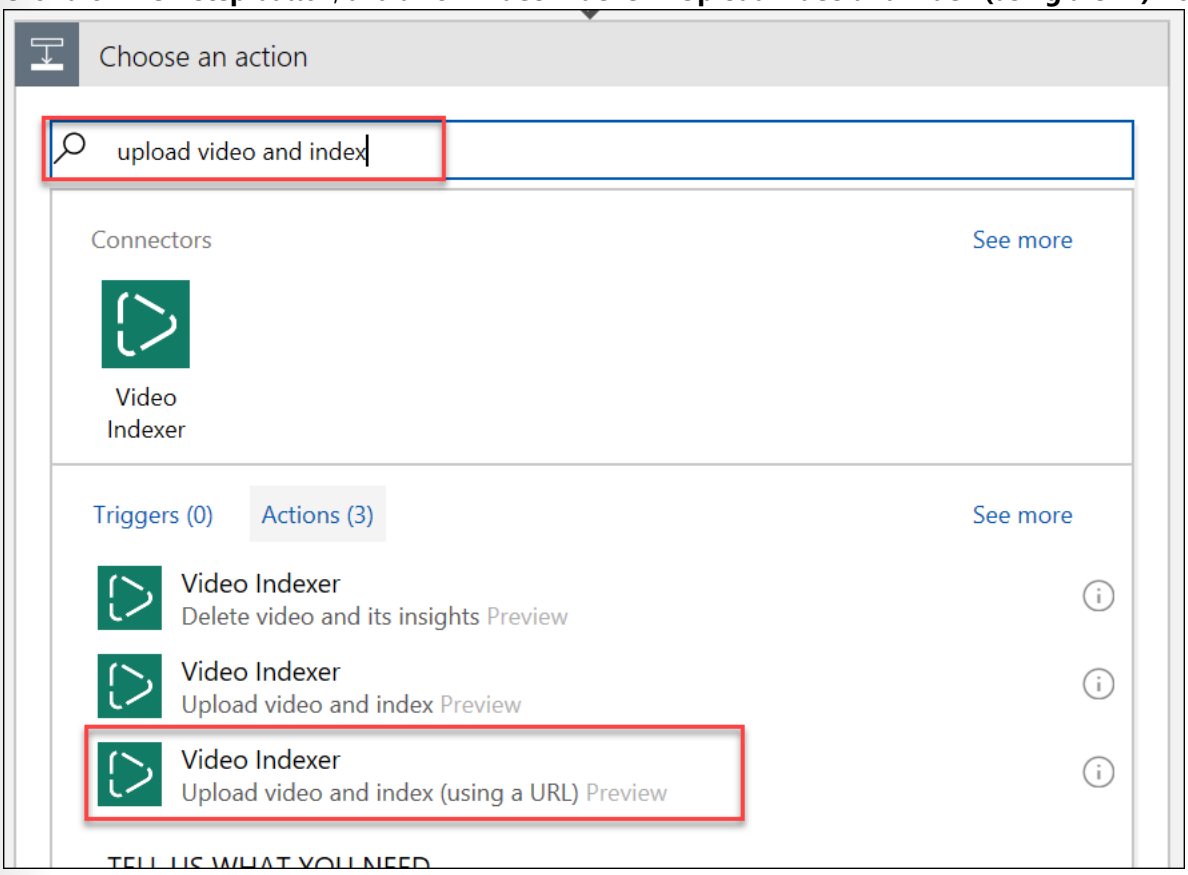
12. Search for and add the **Azure Blob Storage – Create SAS URI by path** action



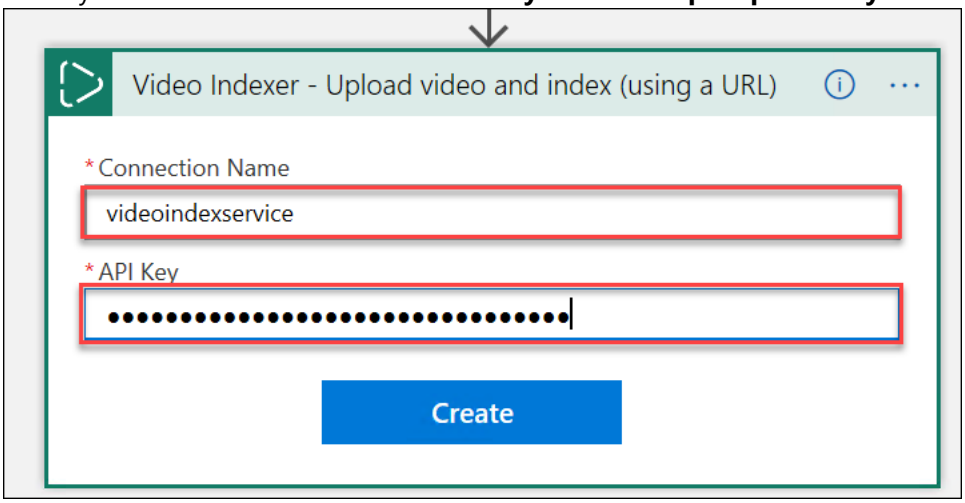
13. Click on the **Blob path** field, then choose to insert the **Path** parameter into the field from the options that display. This will use the Path of the Blob from the Trigger as the value to use when creating the SAS URI in this Action.



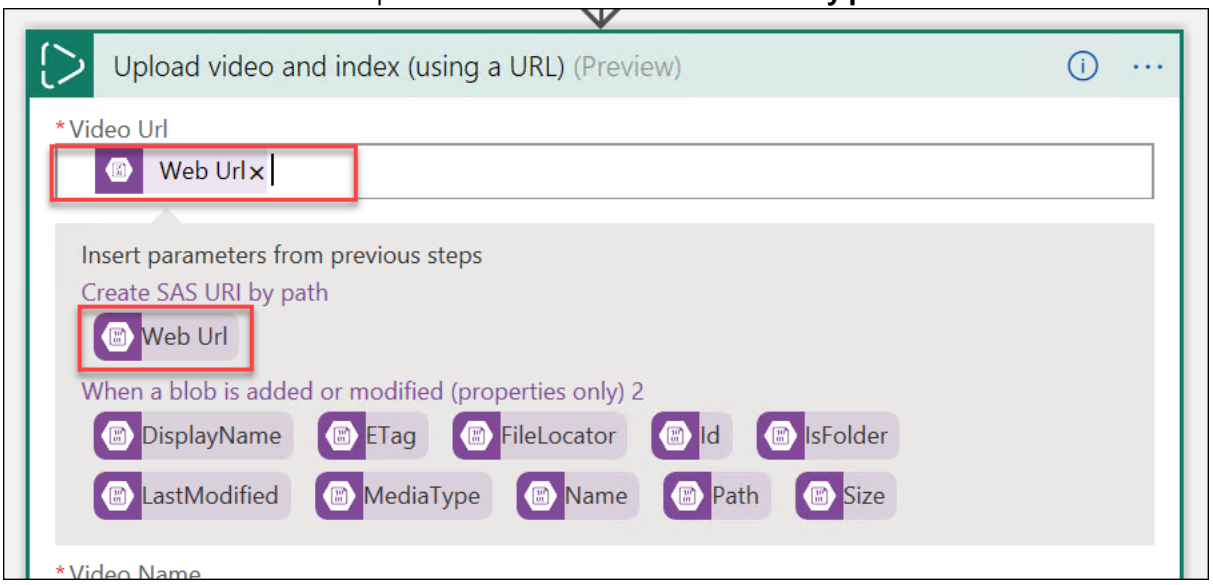

14. Click the **+New step** button, and a new **Video Indexer – Upload video and index (using a URL)** Action



15. Enter the following values for the **Video Indexer – Upload video and index (using a URL)** connection, then click **Create**
    a. Connection Name: **enter a name for the connection**
    b. API Key: **Paste in the Video Indexer API Key that was copied previously**



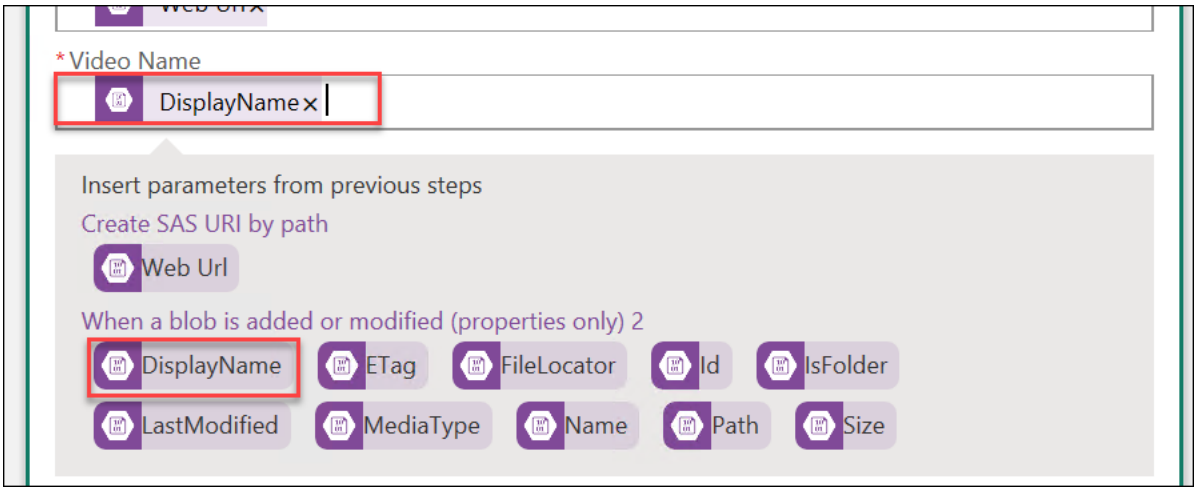

16. Enter the following values for the **Video Indexer – Upload video and index (using a URL)** action:
    a. Video Url: Choose the **Web Url** parameter from the **Create SAS URI by path** action



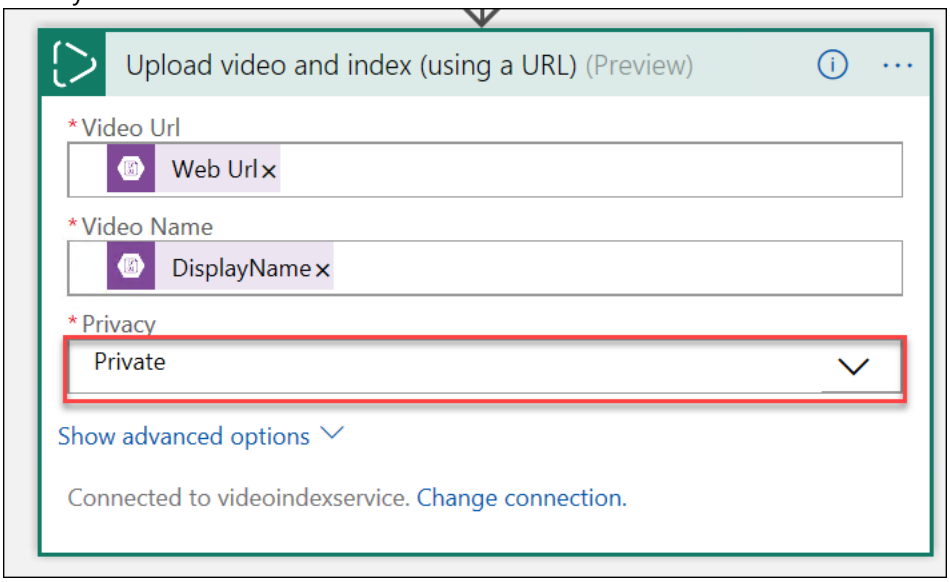


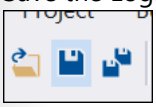
b. Video Name: Choose the **DisplayName** parameter from the **When one or more blobs are added or modified** action



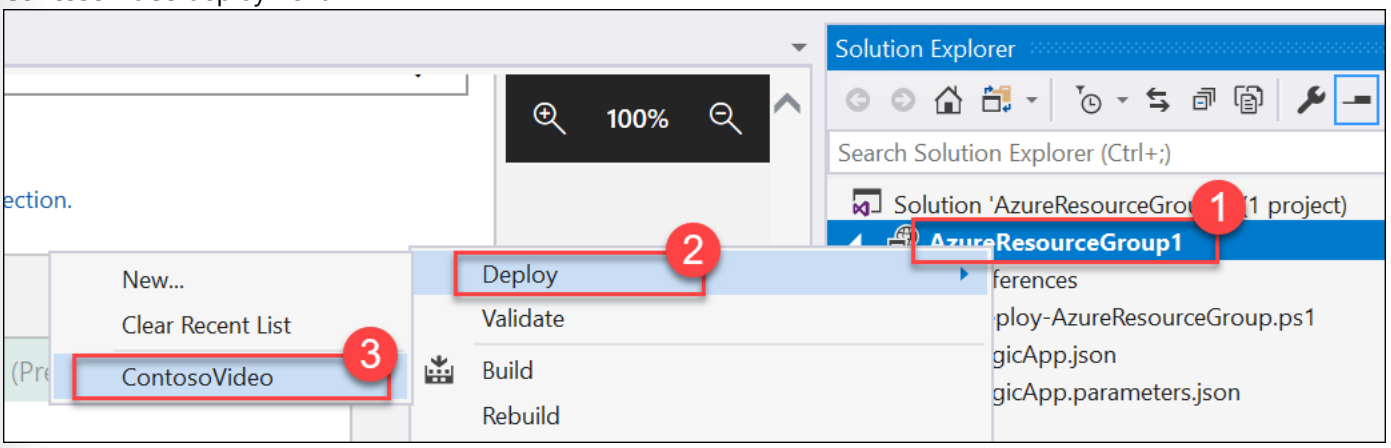

c. Privacy: **Private**



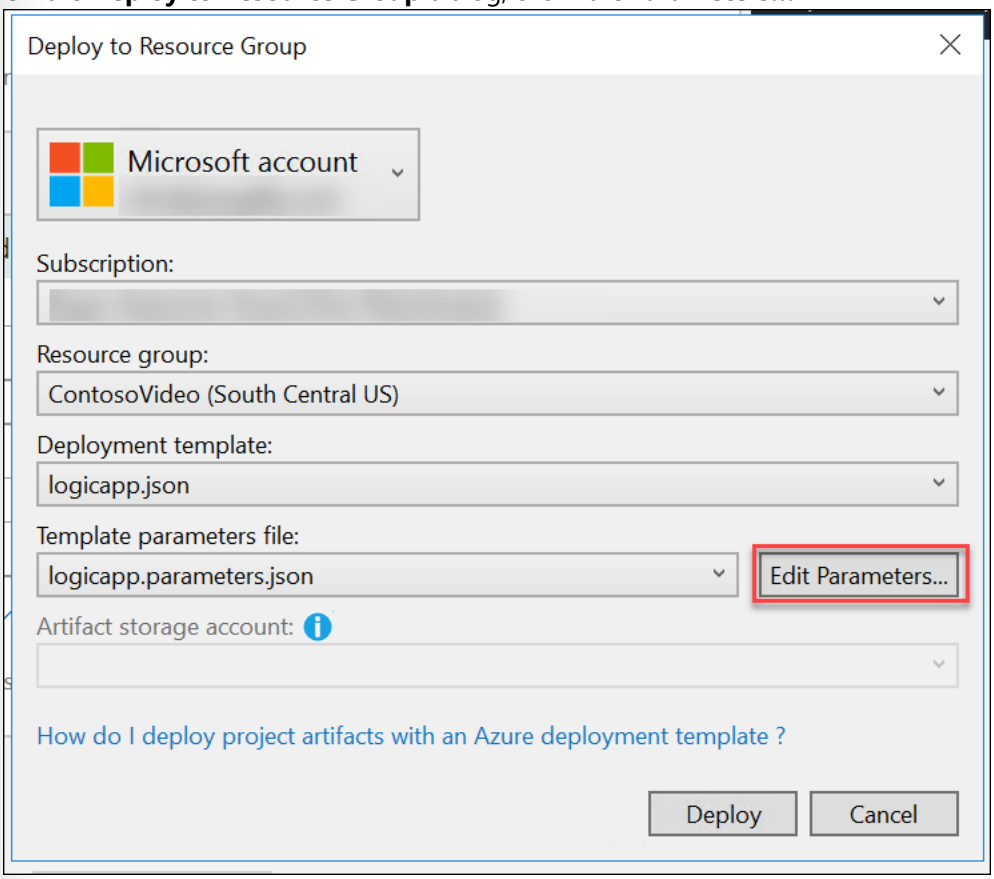

17. Save the Logic App



18. Within the **Solution Explorer** pane, right-click the **Resource Group Project**, click **Deploy**, then click on the **ContosoVideo** deployment



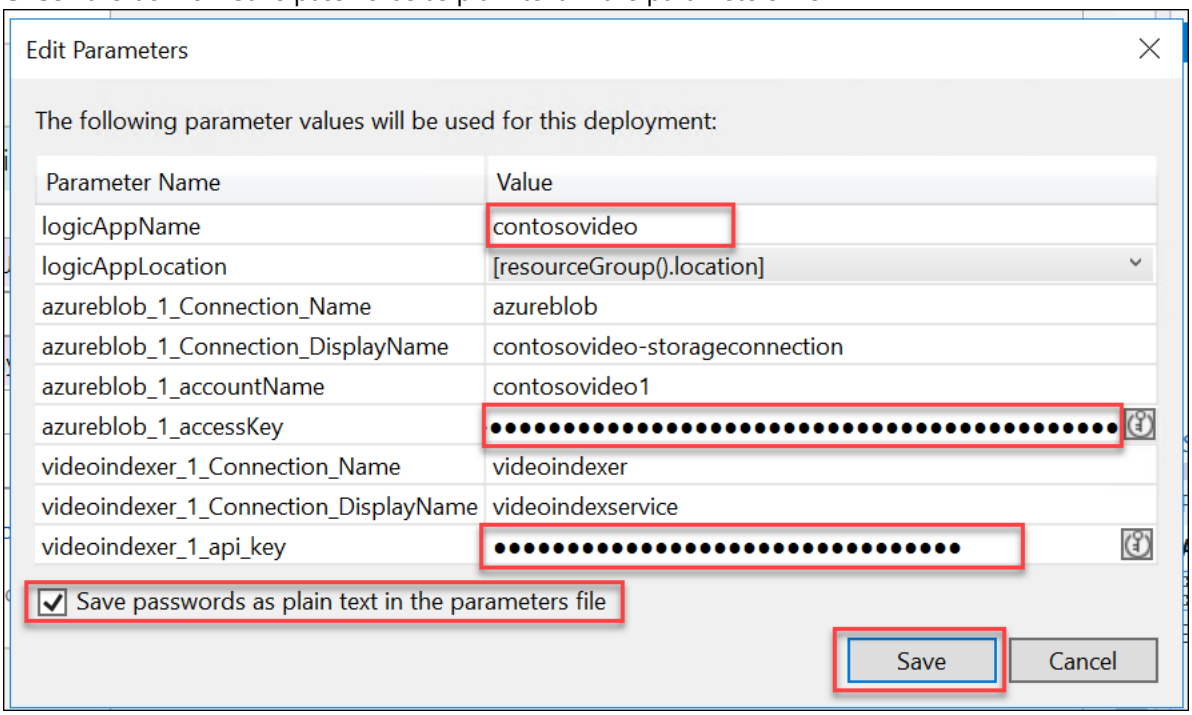

19. On the **Deploy to Resource Group** dialog, click **Edit Parameters...**



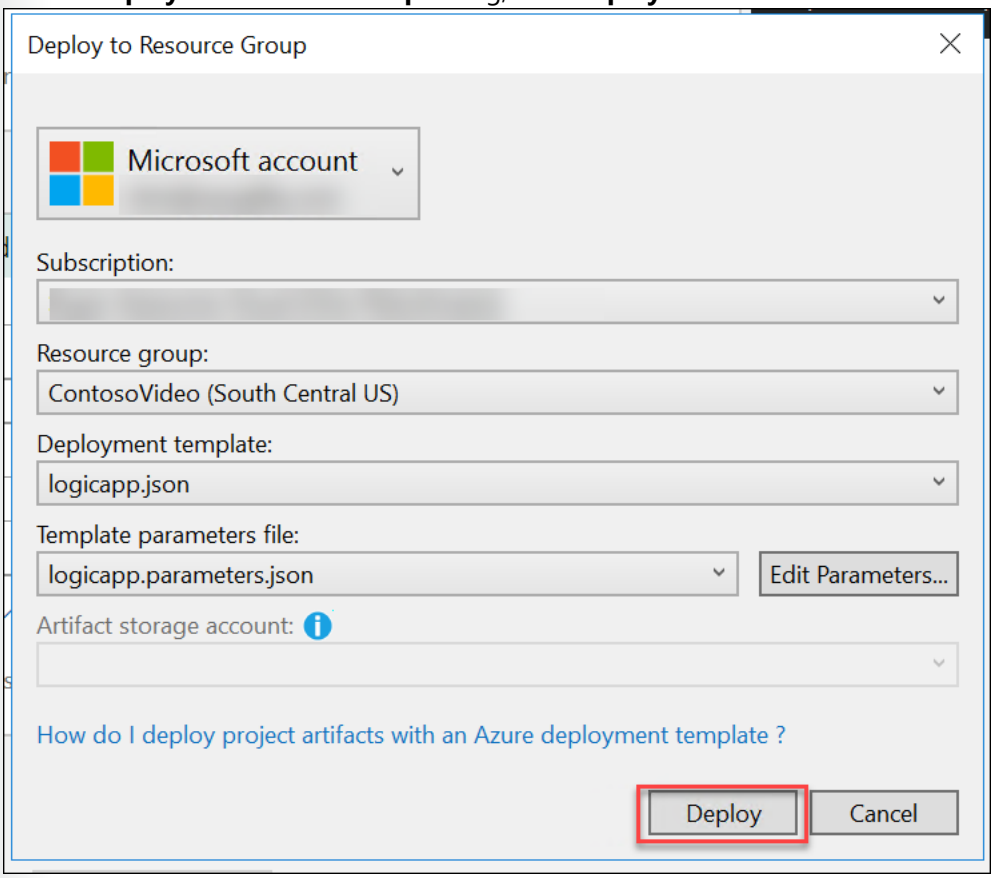

20. On the **Edit Parameters** dialog, enter the following values, then click **Save**:
    a. logicAppName: **enter a unique name for the Logic App**
    b. azureblob_1_accessKey: **paste in the Key 1 that was copied from the Storage Account previously**

c. videoindexer_1_api_key: **paste in the API Key for the Video Indexer service**

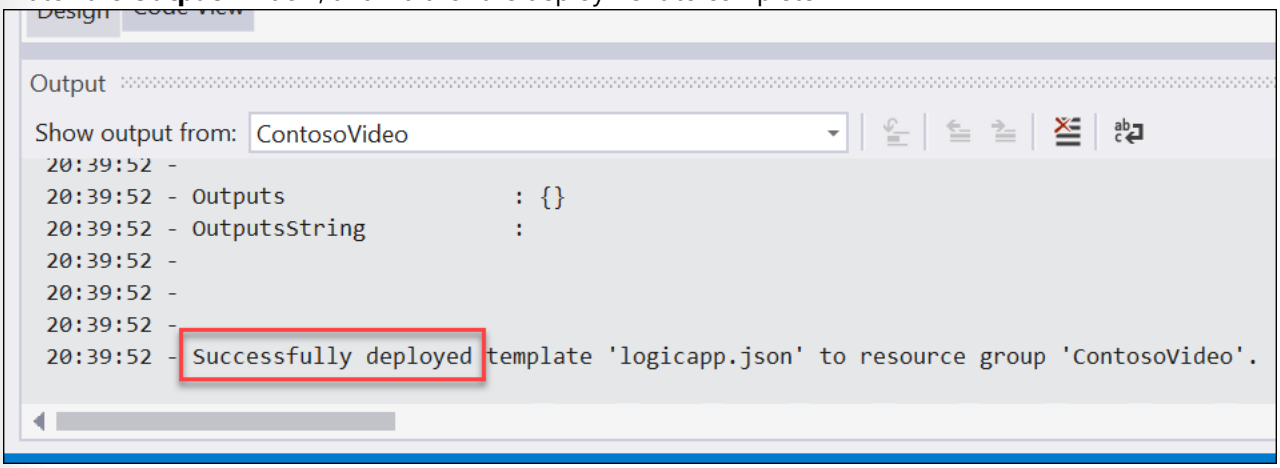
d. **Check** the box for "Save passwords as plain text in the parameters file"




21. On the **Deploy to Resource Group** dialog, click **Deploy**



22. Watch the **Output** window, and wait for the deployment to complete





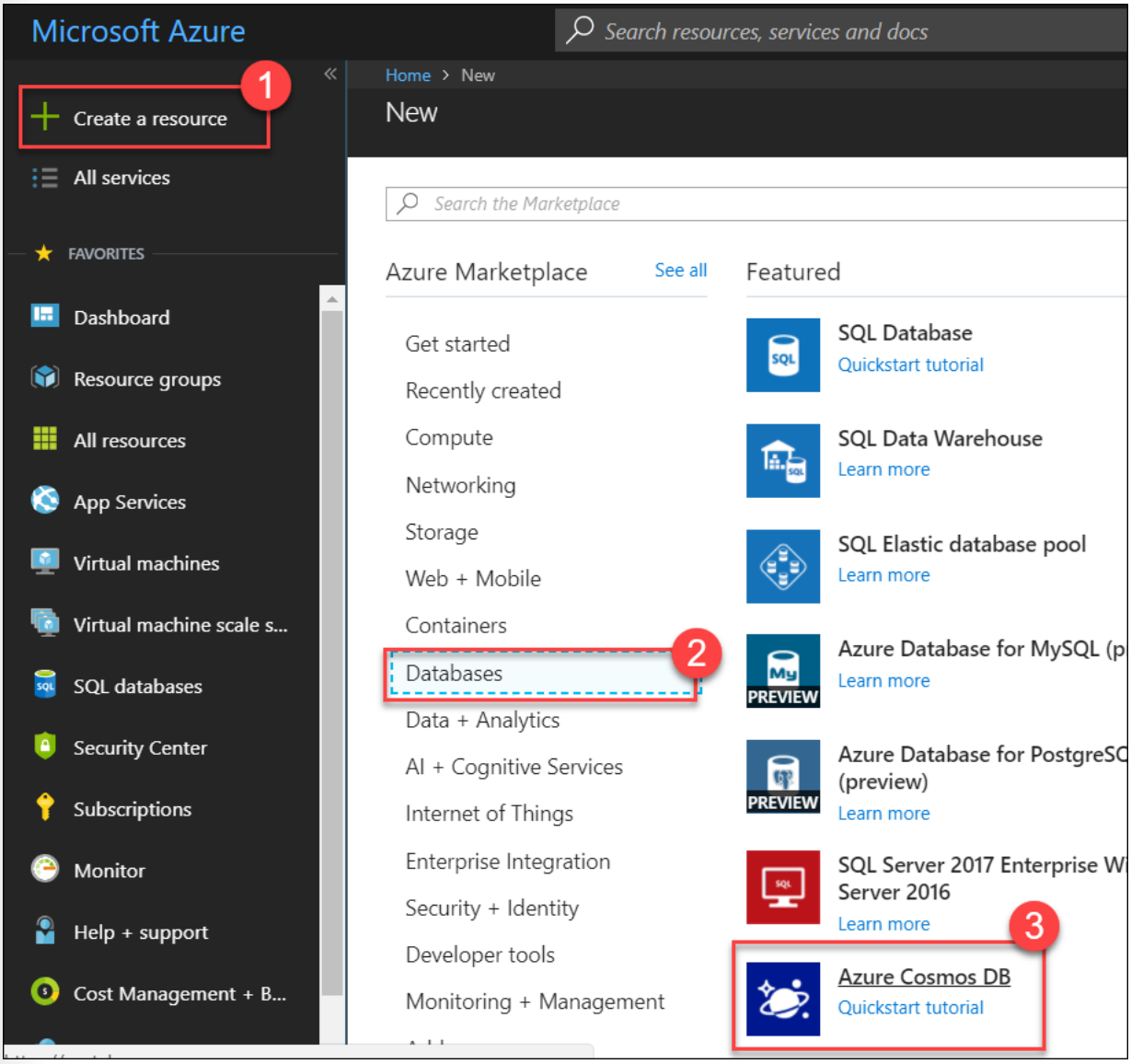
# Exercise 3: Enable admin website to upload videos

Duration: 45 minutes

In this exercise, you will wire up the Admin website to enable Video Upload functionality.

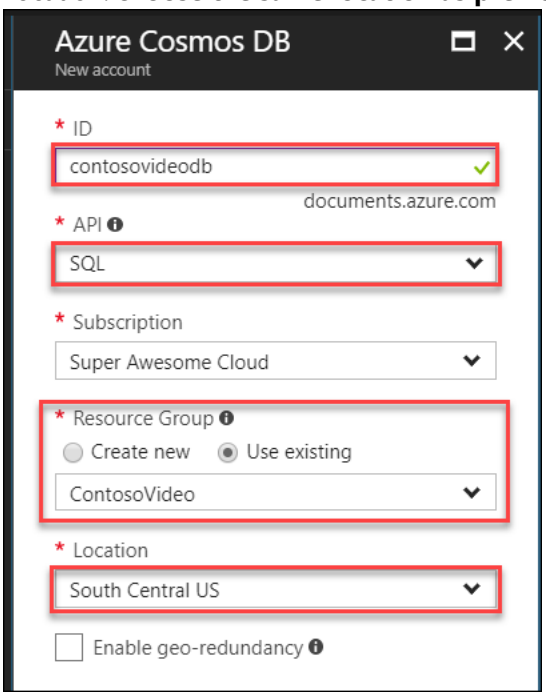
## Task 1: Provision Cosmos DB Account

1. In the left-side menu, click **+Create a resource**, then **Databases**, then **Azure Cosmos DB**



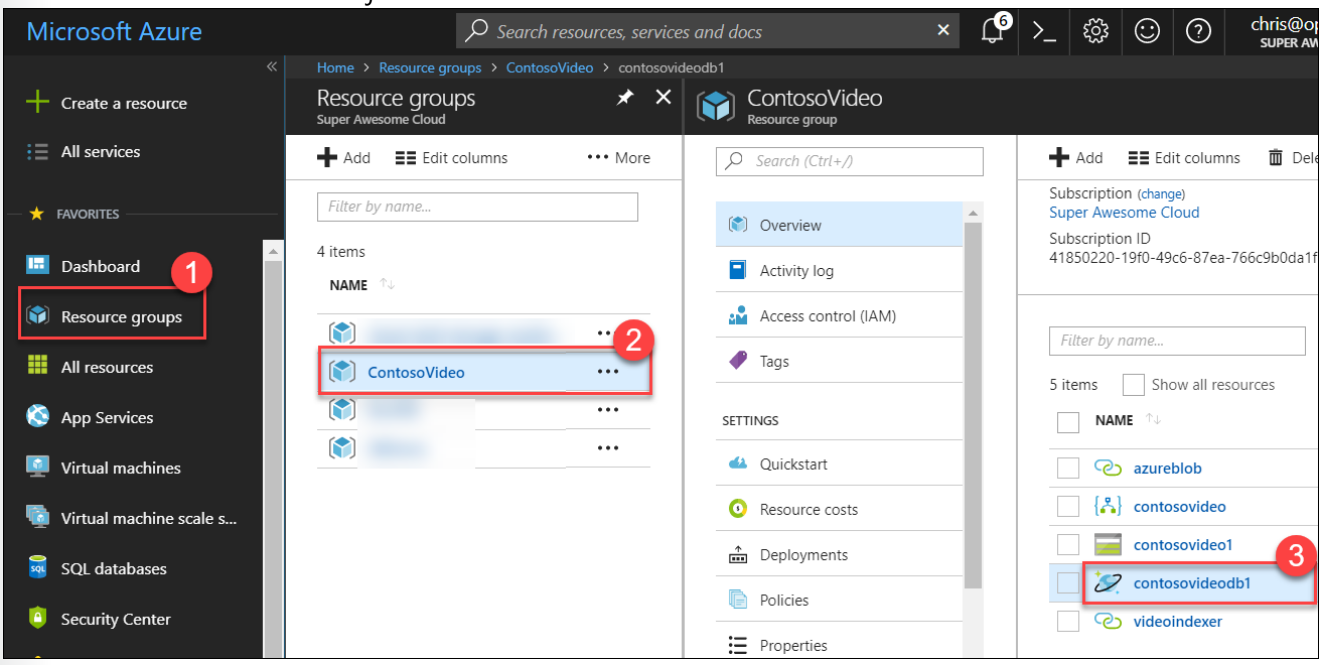

2. On the **Azure Cosmos DB New account** blade, enter the following values:
   a. ID: **enter a unique name**
   b. API: **SQL**
   c. Resource Group: **ContosoVideo**

d. Location: **choose the same location as previously**

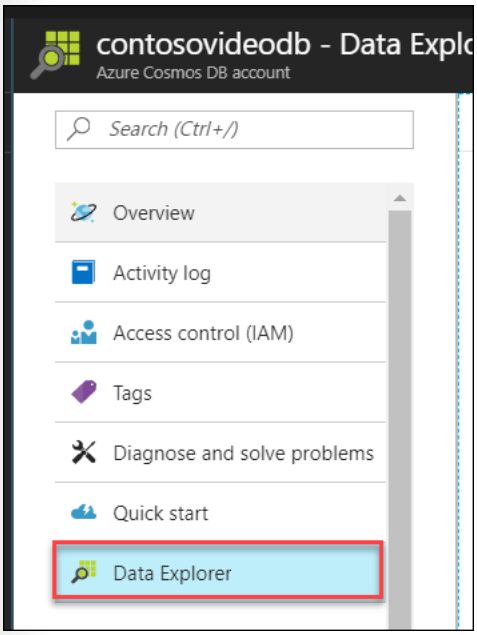



3. Click **Create**

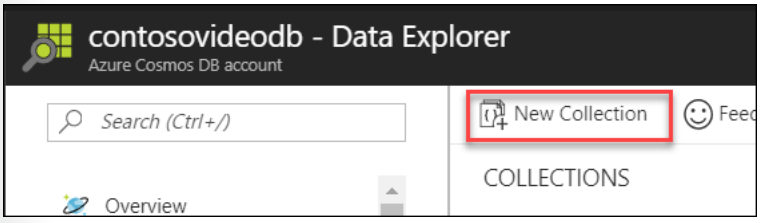
4. In the left-side menu, click **Resource groups**, then click the **ContosoVideo** Resource group, then click on the **Cosmos DB Account** that was just created



5. On the **Cosmos DB Account** blade, click on **Data Explorer**

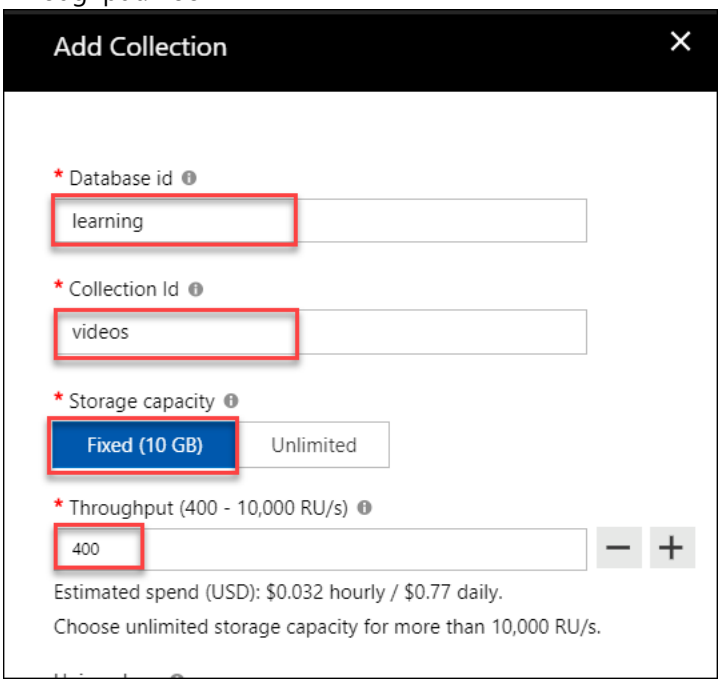



6. Click the **New Collection** button




7. On the **Add Collection** pane, enter the following values:
   a. Database id: **learning**
   b. Collection Id: **videos**
   c. Storage capacity: **Fixed (10GB)**

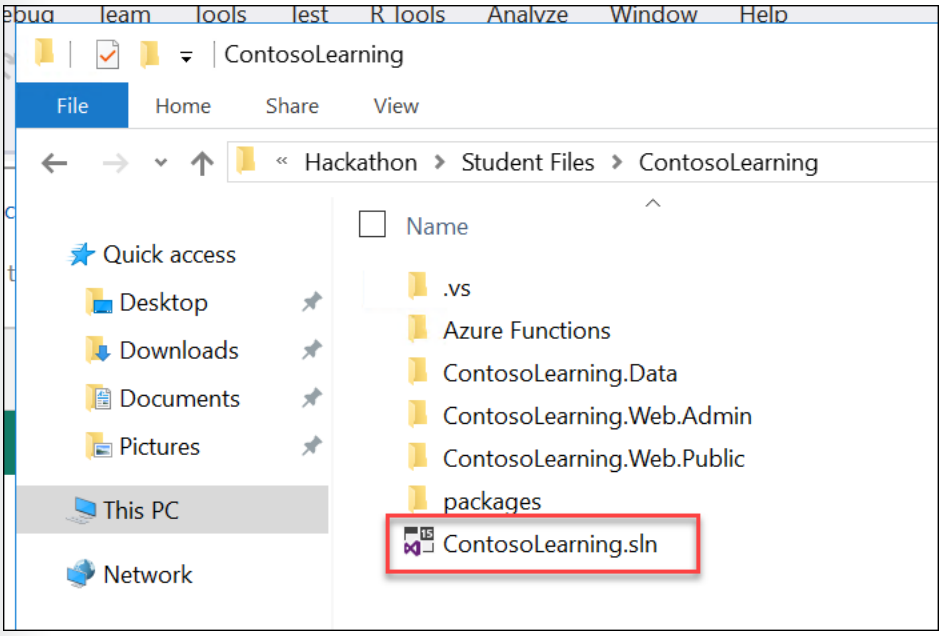
d. Throughput: **400**



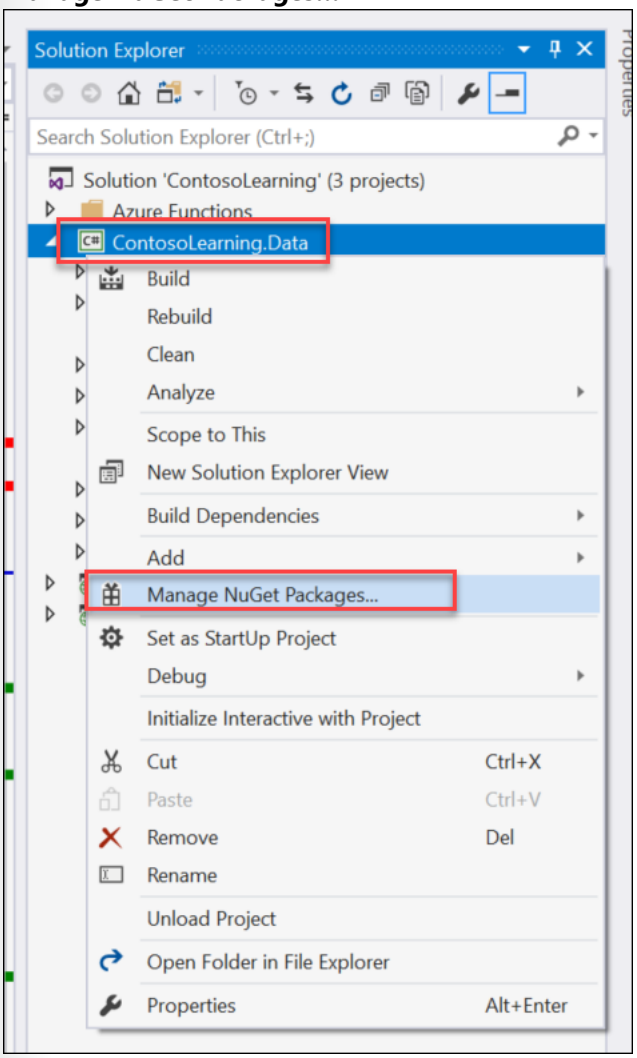

8. Click **OK**

## Task 2: Integrate Cosmos DB into Admin Web App

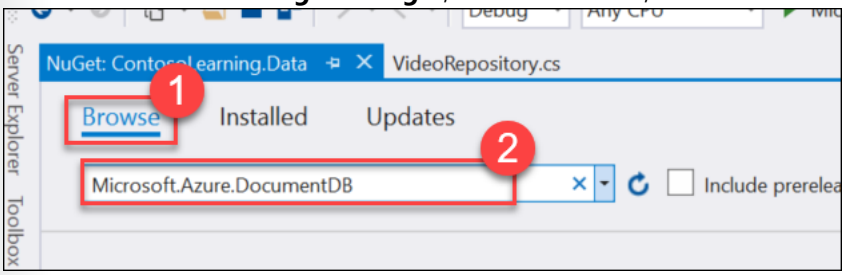
1. Within the folder where the exercise files have been extracted to, Open the **ContosoLearning.sln** solution within Visual Studio 2017



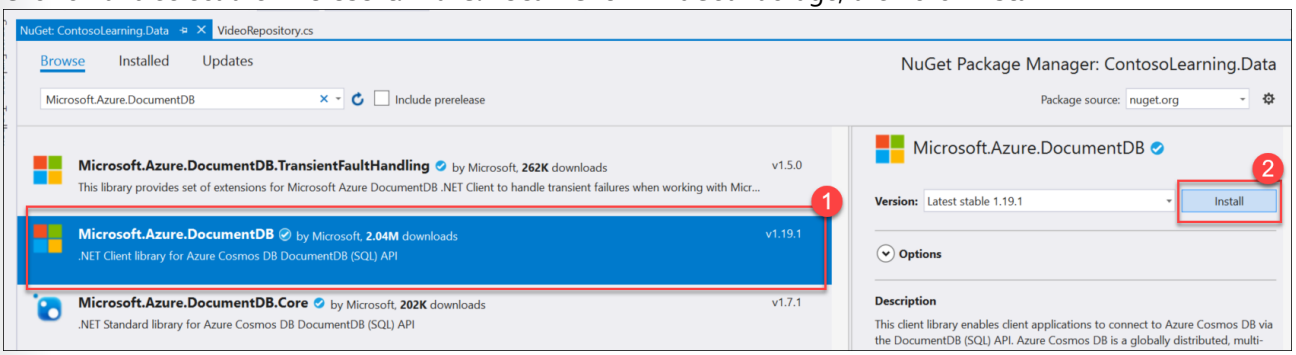
2. Within Solution Explorer, locate the **ContosoLearning.Data** project, then right-click the project and click on **Manage NuGet Packages...**



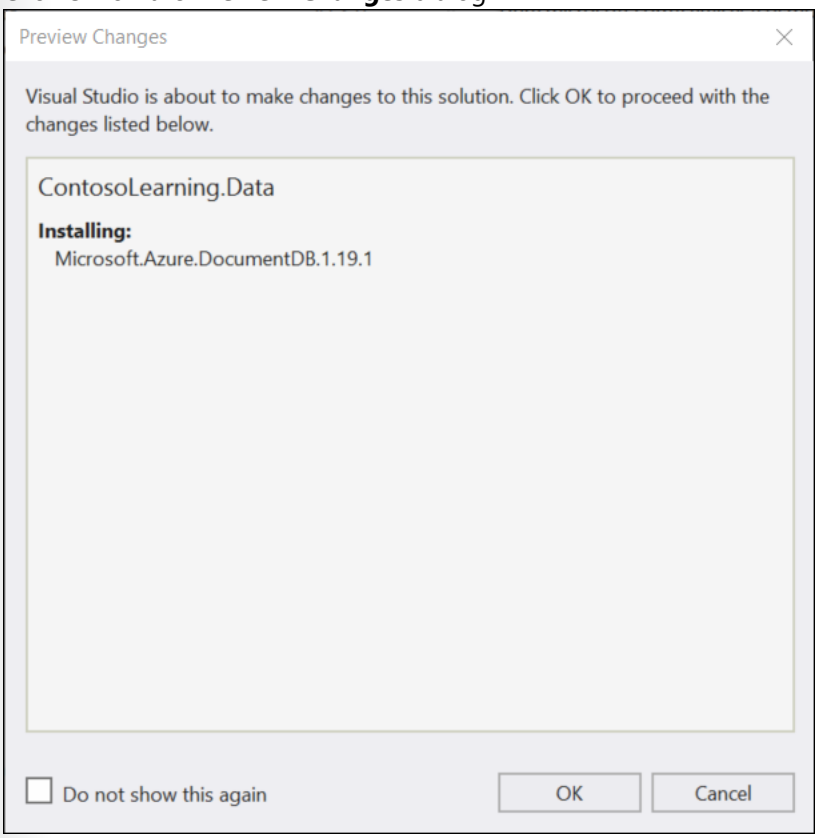

3. Within the **NuGet Package Manager**, click on **Browse**, then search for **Microsoft.Azure.DocumentDB**



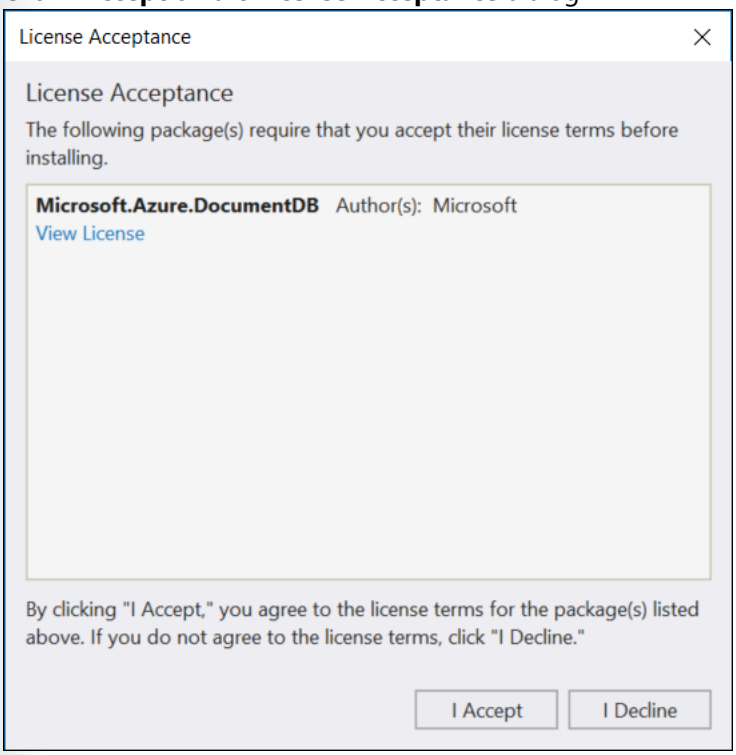
4. Click on and select the **Microsoft.Azure.DocumentDB** NuGet Package, then click **Install**



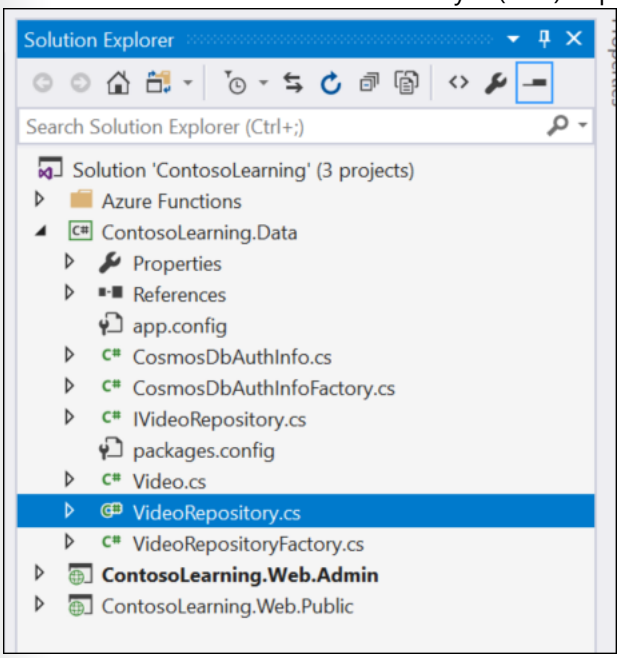

5. Click **OK** on the **Preview Changes** dialog



6. Click **I Accept** on the **License Acceptance** dialog




7. Within Solution Explorer, locate the **ContosoLearning.Data** project, and open the **VideoRepository.cs** file. This is the source code file that will contain the source code for interactive with the Cosmos DB Database / Collection within this solution. This data access layer (DAL) implementation uses a simplified Repository pattern.




8. Add the following **using** statements to the top of the **VideoRepository.cs** file

```
using Microsoft.Azure.Documents.Client;
using Microsoft.Azure.Documents.Linq;
```

9. Add the following method named **createDocumentClient()** to the **VideoRepository** class. This is a reusable method that will be used by a couple different methods within the class to instantiate a new DocumentClient class instance for working with Cosmos DB.

```
private DocumentClient createDocumentClient()
{
    return new Microsoft.Azure.Documents.Client.DocumentClient(
        new Uri(_cosmosDbAuthInfo.Endpoint),
        _cosmosDbAuthInfo.AuthKey
        );
}
```

10. Modify the code for the **GetAll()** method to the following to load all Documents from Cosmos DB and return them as a List object

```
var list = new List<Video>();

using (var documentClient = this.createDocumentClient())
{
    var documentQuery = documentClient.CreateDocumentQuery<Video>(
        UriFactory.CreateDocumentCollectionUri(_cosmosDbAuthInfo.Database,
_cosmosDbAuthInfo.Collection)
        ).AsDocumentQuery();

    while (documentQuery.HasMoreResults)
    {
        list.AddRange(
            await documentQuery.ExecuteNextAsync<Video>()
            );
    }
}

return list;
```

11. Notice that the above code references a **_cosmosDbAuthInfo** object. This object is prepopulated with the necessary values for the Cosmos DB Account and Collection by loading from **AppSettings**. These AppSettings will be populated later.

12. Replace the body of the **Get(string id)** method with the following code that will retrieve a single Document by ID

```
using (var documentClient = this.createDocumentClient())
{
    var response = await documentClient.ReadDocumentAsync<Video>(
        UriFactory.CreateDocumentUri(_cosmosDbAuthInfo.Database, _cosmosDbAuthInfo.Collection, id)
        );

    return response.Document;
}
```

13. Replace the body of the **Delete(string videoId)** method with the following code that will delete a single Document by ID

```
using (var documentClient = this.createDocumentClient())
{
    await documentClient.DeleteDocumentAsync(
        UriFactory.CreateDocumentUri(_cosmosDbAuthInfo.Database, _cosmosDbAuthInfo.Collection,
videoId)
        );
}
```

14. Add the following code to the **Insert(Video video)** method in place of the *"// Code Here"* comment. This code will insert a new Document into the Cosmos DB Collection

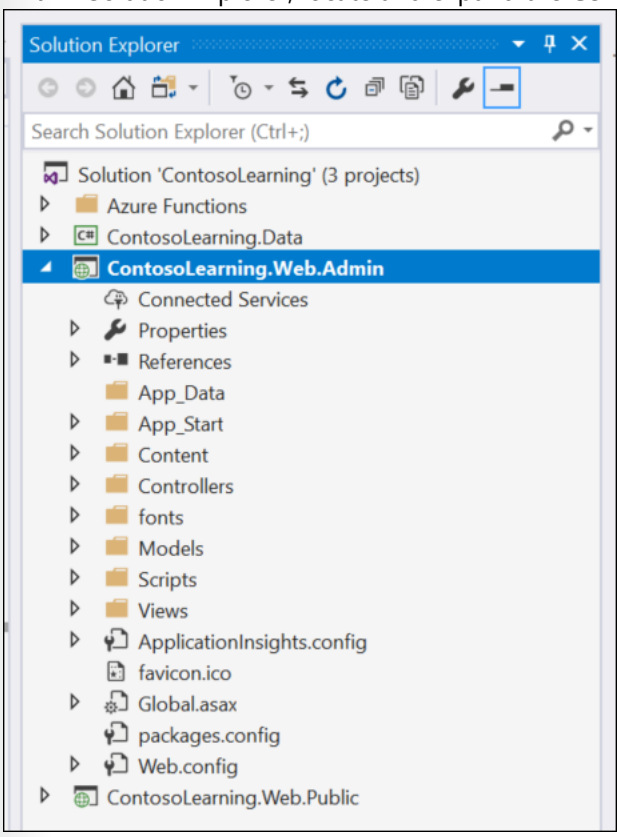
```
using (var documentClient = this.createDocumentClient())
{
    await documentClient.CreateDocumentAsync(
        UriFactory.CreateDocumentCollectionUri(_cosmosDbAuthInfo.Database,
_cosmosDbAuthInfo.Collection),
        Video
        );
}
```

15. Save the file

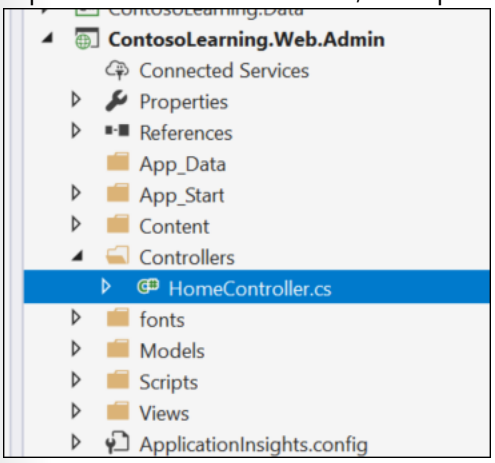
## Task 3: Integrate File Upload into Admin Web App

1. Open the **ContosoLearning.sln** solution within Visual Studio 2017

2. Within Solution Explorer, locate and expand the **ContosoLearning.Web.Admin** project




3. Expand the **Controllers** folder, and open the **HomeController.cs** file




4. Add the following using statements to the top of the **HomeController.cs** file

```
using Microsoft.WindowsAzure.Storage;
using System.Configuration;
```

5. Within the **HomeController**, locate the **Upload** method with the **HttpPost** attribute

```
[HttpPost]
public async Task<ActionResult> Upload(HttpPostedFileBase file, HomeUploadModel model)
{
    if (!ModelState.IsValid)
    {
        return View(model);
    }

    // Generate a unique id for the uploaded video
    var videoId = Guid.NewGuid().ToString();

    // =====================================================================
    // Upload Video File to Blob Storage
    // =====================================================================

    // Code Here


    // =====================================================================
    // Save Video info to Cosmos DB Database / Collection
    // =====================================================================

    // Code Here


    return RedirectToAction("Index");
}
```

6. Paste in the following code where the comment states *"Upload Video File to Blob Storage"*. This code will take the file uploaded to the Action Method via an HTTP Post, and upload that file to the Blob Storage Account.

```
// Load Connection String to Azure Storage Account
var videoConnString = ConfigurationManager.ConnectionStrings["videostorage"].ConnectionString;
if (string.IsNullOrWhiteSpace(videoConnString))
{
    throw new Exception("The 'videostorage' Connection String is NOT set");
}

// Get reference to the Blob Container to upload to
var storageAccount = CloudStorageAccount.Parse(videoConnString);
var blobClient = storageAccount.CreateCloudBlobClient();
var container = blobClient.GetContainerReference("video");
await container.CreateIfNotExistsAsync();

// Upload Video file to Blob Storage
var blob = container.GetBlockBlobReference(videoId);
await blob.UploadFromStreamAsync(file.InputStream);
```

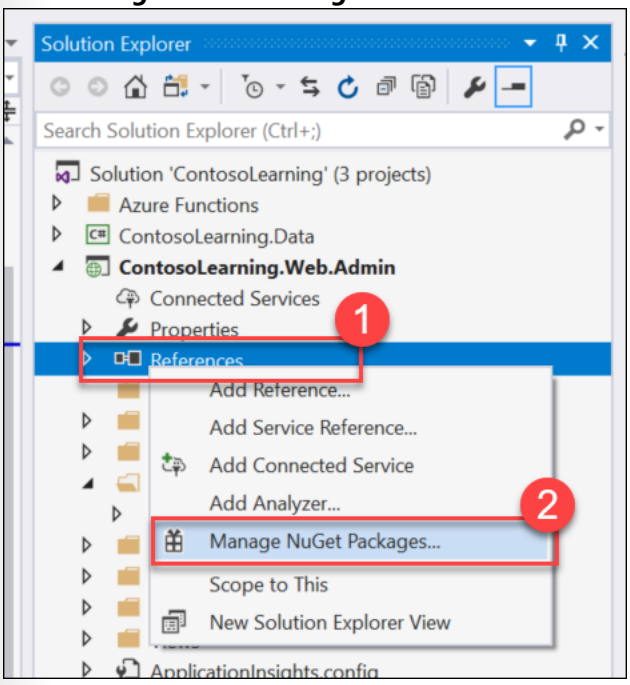
7. Paste in the following code where the comment states *"Save Video info to Cosmos DB..."*. This code will use the VideoRepository to save a new Document to Cosmos DB for the newly uploaded video.

```
// Save new Document to Cosmos DB for this Video
var videoRepo = VideoRepositoryFactory.Create();
var video = new Video();
video.Id = videoId;
video.Title = model.Title;
video.Description = model.Description;
video.ProcessingState = "Processing";
video.ProcessingProgress = "0%";
await videoRepo.Insert(video);
```

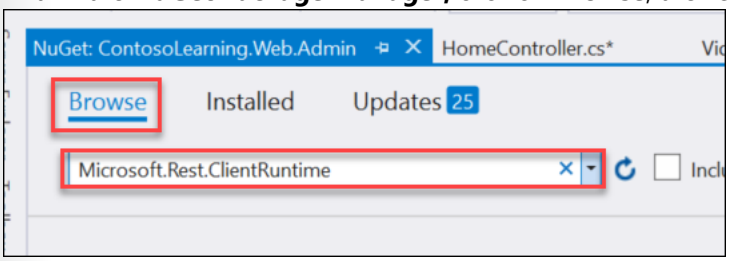
8. Save the file

## Task 4: Add ability to delete video

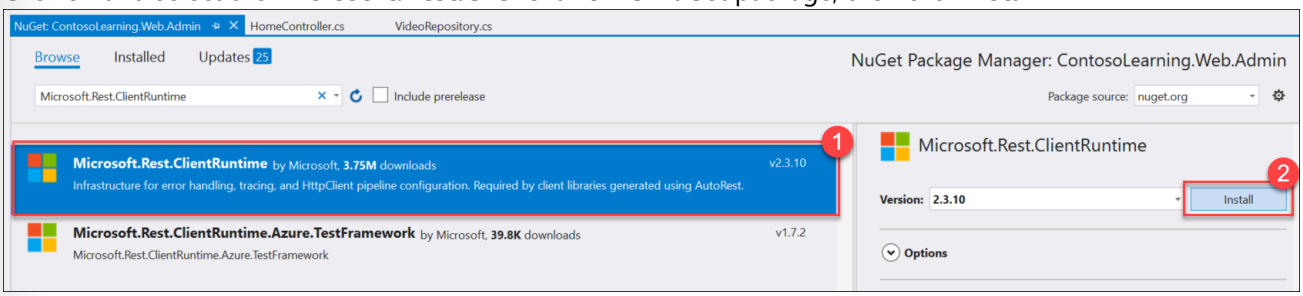
1. Within Solution Explorer, expand the **ContosoLearning.Web.Admin** project, then right-click on **References,** then click **Manage NuGet Packages...**



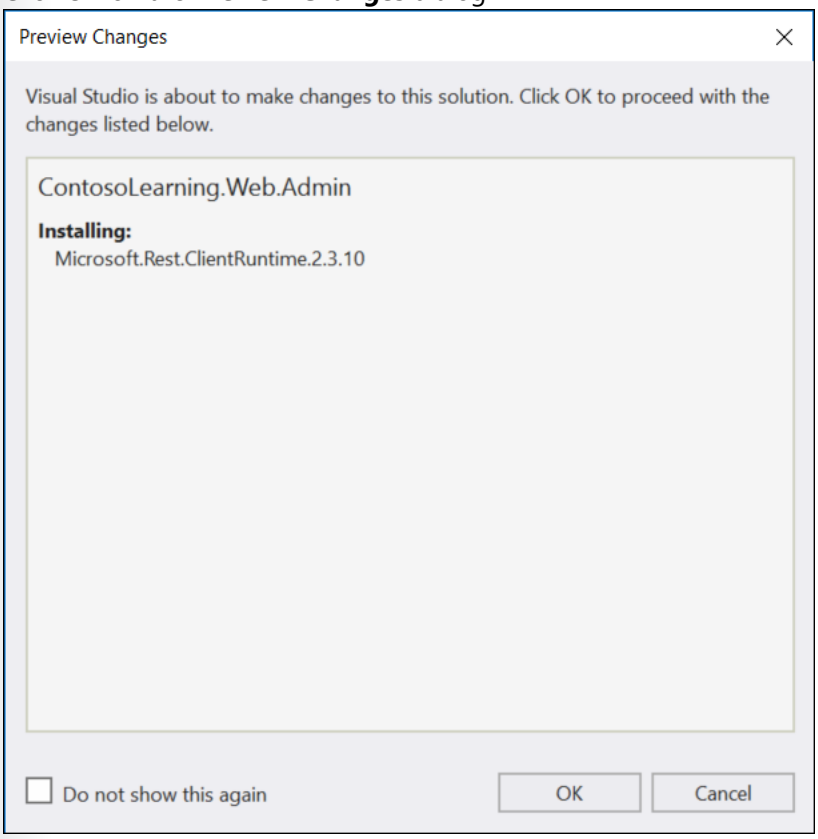

2. Within the **NuGet Package Manager,** click on **Browse**, then search for **Microsoft.Rest.ClientRuntime**



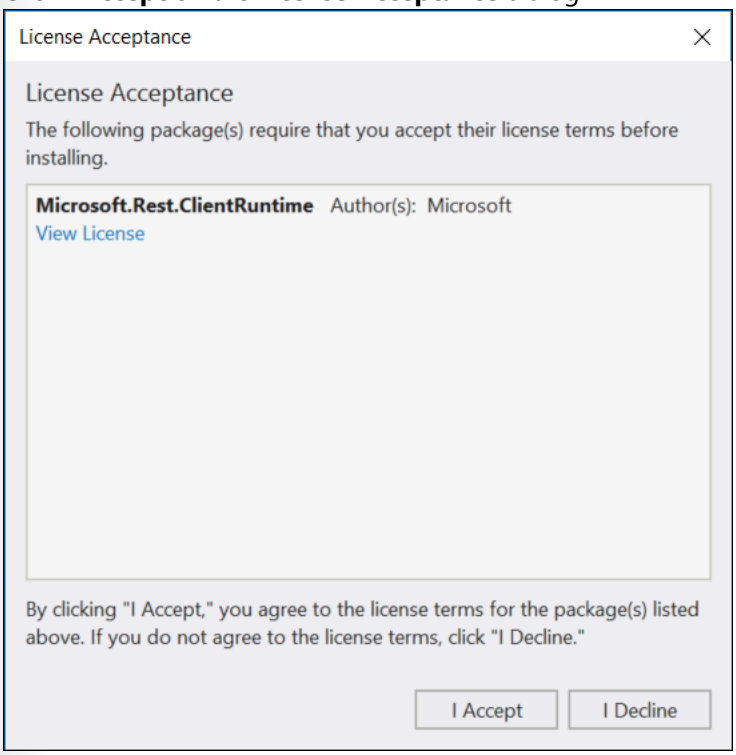
3. Click on and select the **Microsoft.Rest.ClientRuntime** NuGet package, then click **Install**



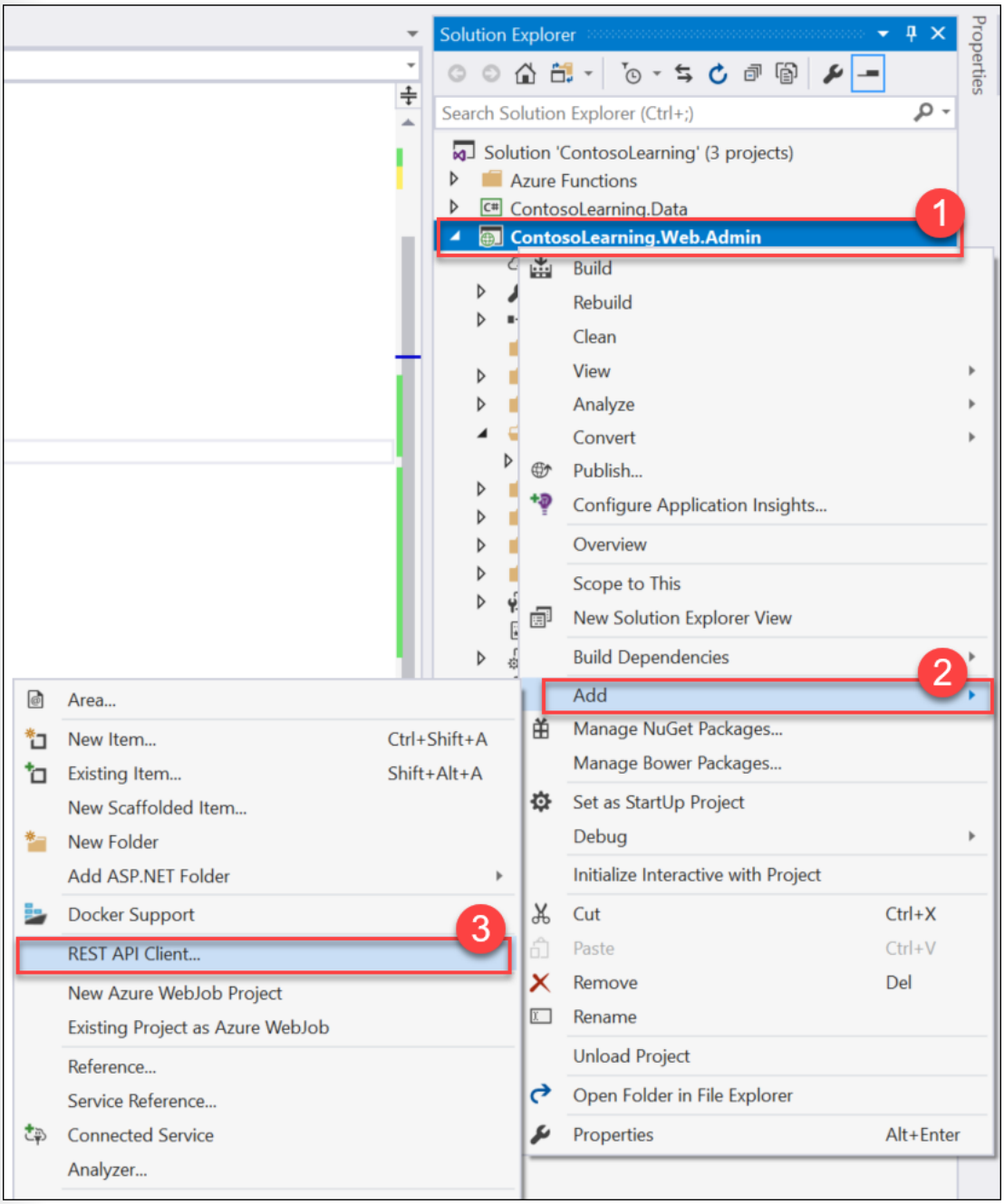

4. Click **OK** on the **Preview Changes** dialog



5. Click **I Accept** on the **License Acceptance** dialog

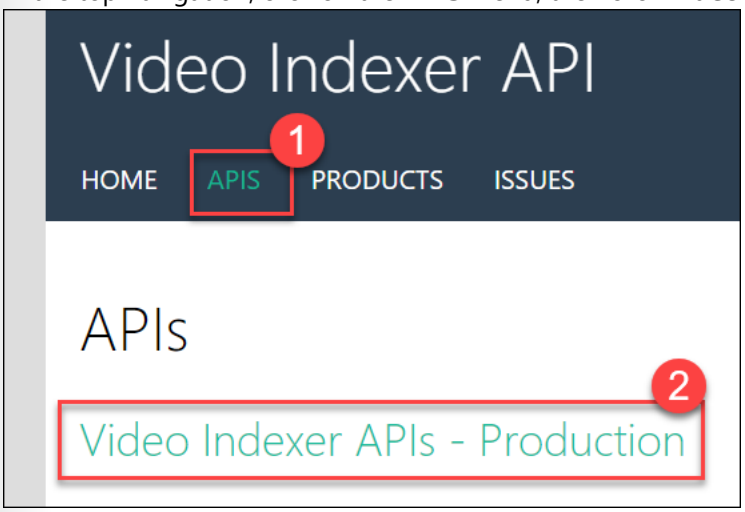


6. Within the Solution Explorer window, right-click the **ContosoLearning.Web.Admin** project, then click **Add**, then **REST API Client...**

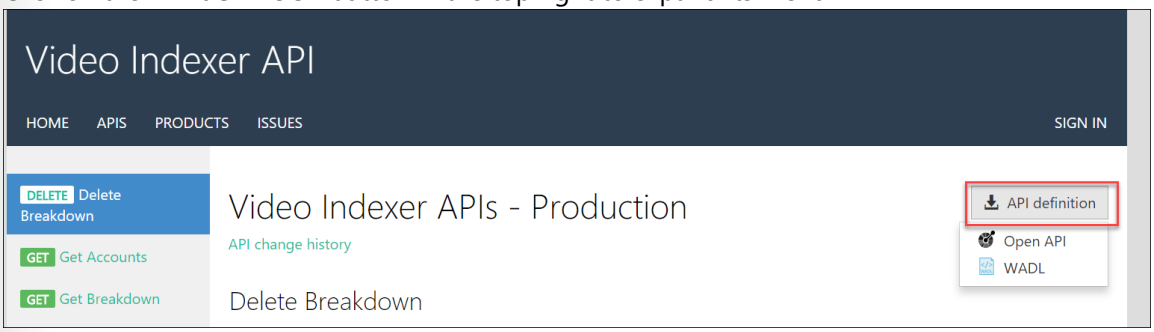



7. Open a new browser window and navigate to the **Video Indexer API** Developer Portal at: https://videobreakdown.portal.azure-api.net/

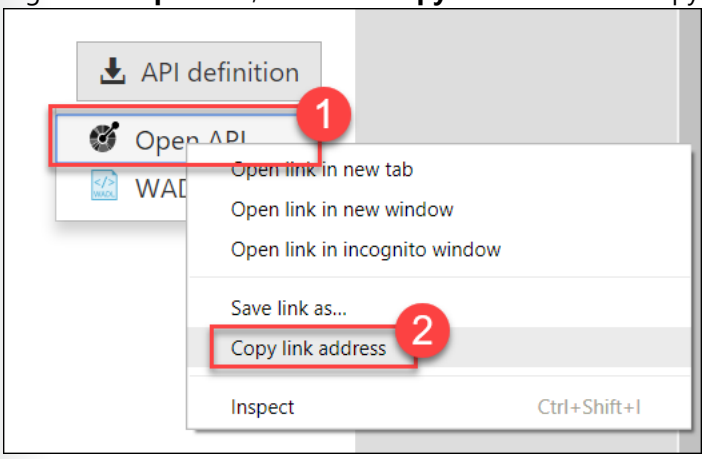
8. In the top navigation, click on the **APIS** menu, then click **Video Indexer APIs – Production**



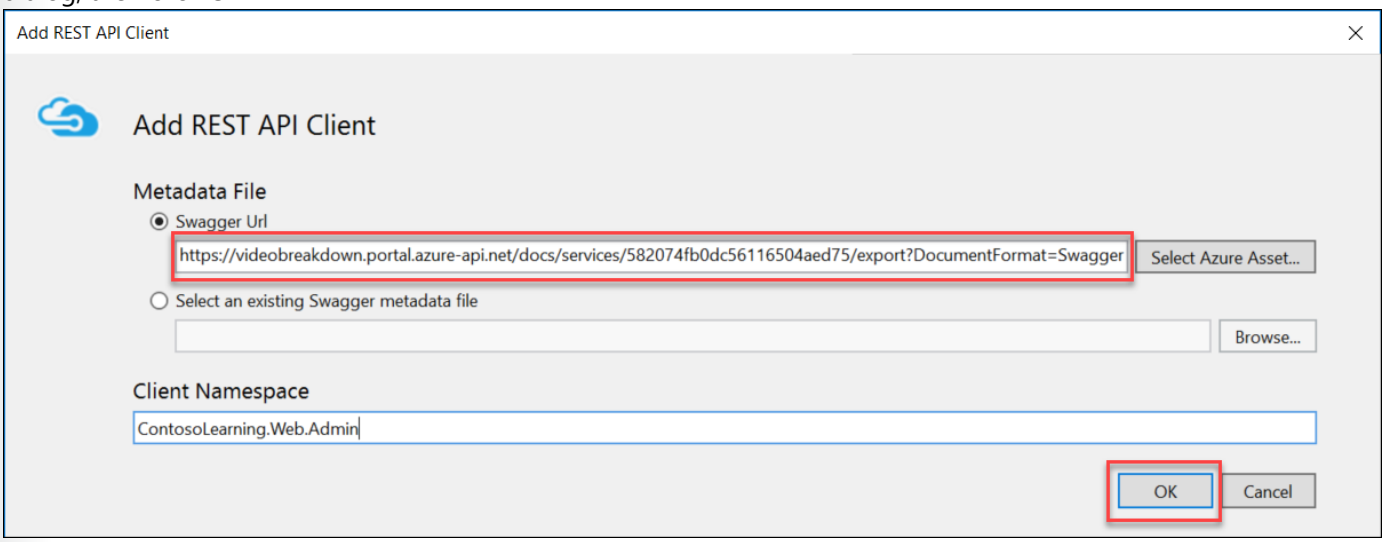

9. Click on the **API definition** button in the top right to expand its menu




10. Right-click **Open API**, then click **Copy link address** to copy the URL for the link to the clipboard



11. Go back to Visual Studio, and paste in the copied **URL** into the **Swagger Url** field of the **Add REST API Client** dialog, then click **OK**




12. Within the **HomeController.cs** file within the **ContosoLearning.Web.Admin** project, locate the **Delete(string id)** Action Method, and replace its contents with the following source code that will delete the Video from both Blob Storage and Cosmos DB

```
// ======================================================================
// Delete document from Cosmos DB
// ======================================================================

var videoRepo = VideoRepositoryFactory.Create();

var video = await videoRepo.Get(id);

await videoRepo.Delete(id);



// ======================================================================
// Delete files from Blob Storage
// ======================================================================

// Load Connection String to Azure Storage Account
var videoConnString = ConfigurationManager.ConnectionStrings["videostorage"].ConnectionString;
if (string.IsNullOrWhiteSpace(videoConnString))
{
    throw new Exception("The 'videostorage' Connection String is NOT set");
}

// Get reference to the Blob Container to upload to
var storageAccount = CloudStorageAccount.Parse(videoConnString);
var blobClient = storageAccount.CreateCloudBlobClient();
```

```
// Get reference to 'video' container
var videoContainer = blobClient.GetContainerReference("video");
await videoContainer.CreateIfNotExistsAsync();

// Delete Video file from Blob Storage
var videoBlob = videoContainer.GetBlockBlobReference(id);
await videoBlob.DeleteAsync();
```

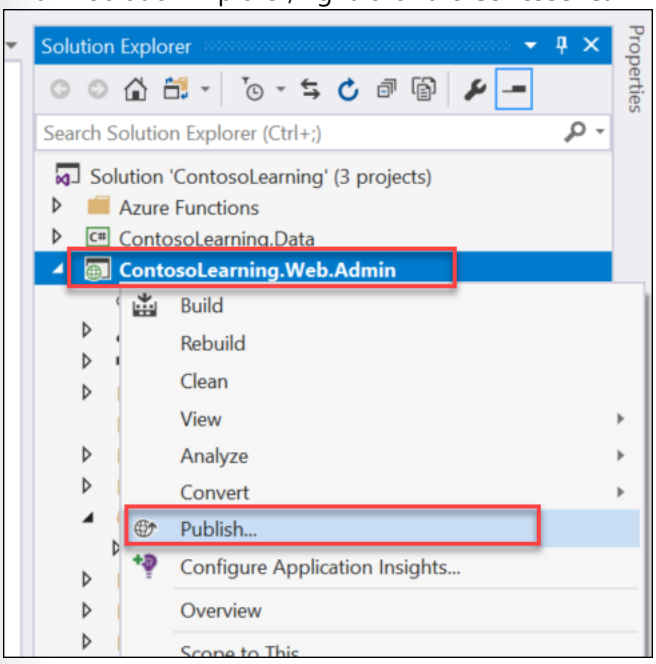
13. After the above code, within the **Delete** method, paste in the following code that will also delete the video from **Video Indexer** when it's deleted from the application.

```
var videoIndexerClient = new AdminClient(
    new Microsoft.Rest.TokenCredentials(
            ConfigurationManager.AppSettings["VideoIndexerAPI_Key"]
        )
    );
var response = await videoIndexerClient.DeleteBreakdownWithHttpMessagesAsync(video.VideoId, true);
```

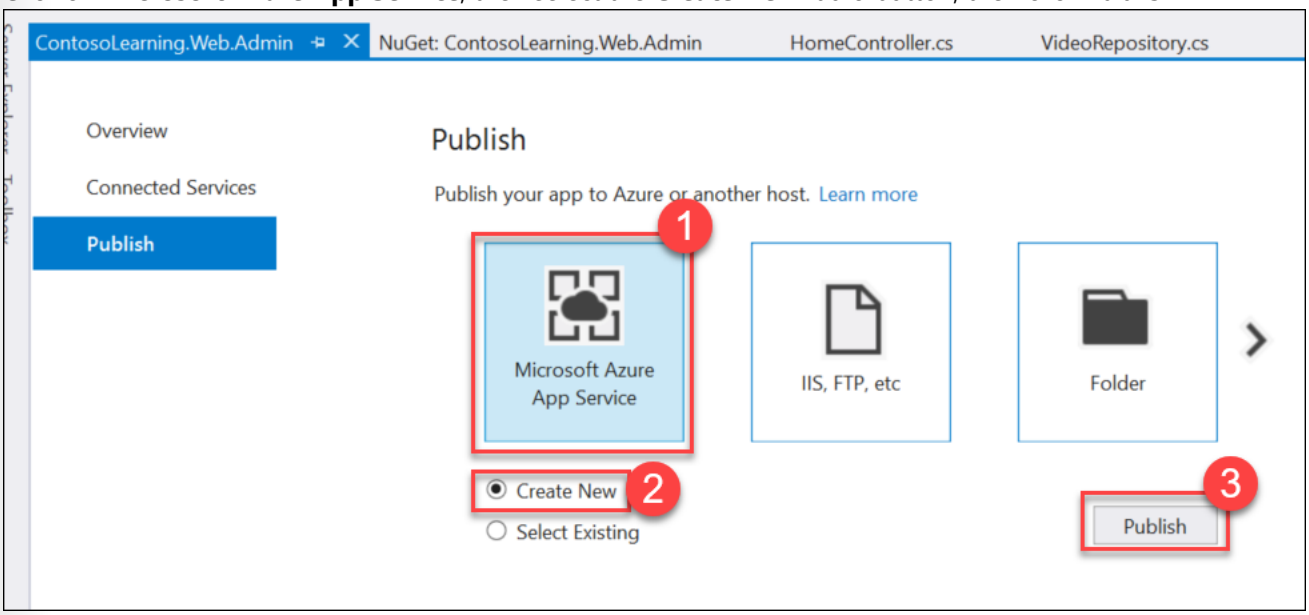
14. Save the file

## Task 5: Deploy Admin website to an Azure Web App

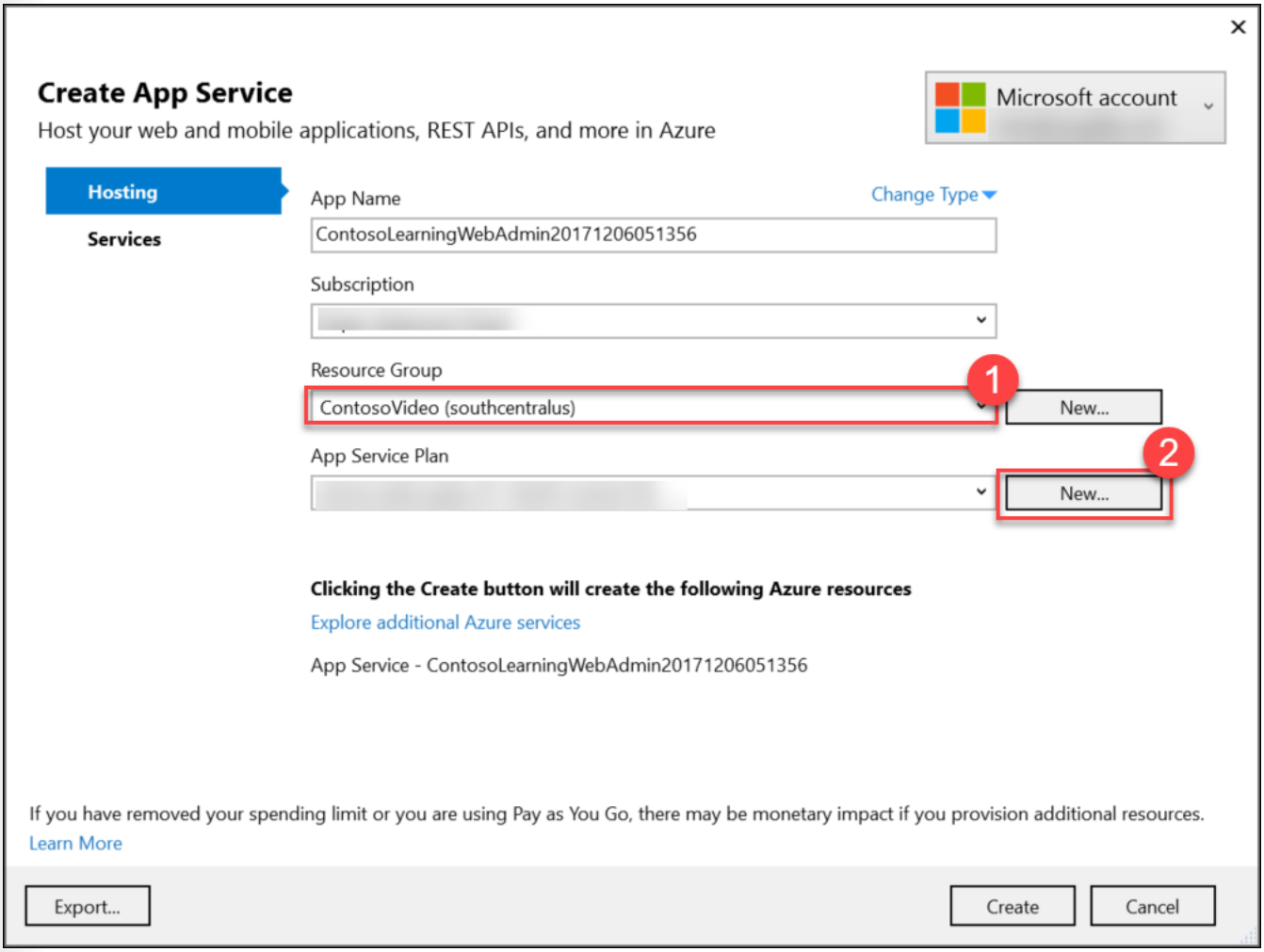
1. Within Solution Explorer, right-click the **ContosoLearning.Web.Admin** project, then click **Publish...**



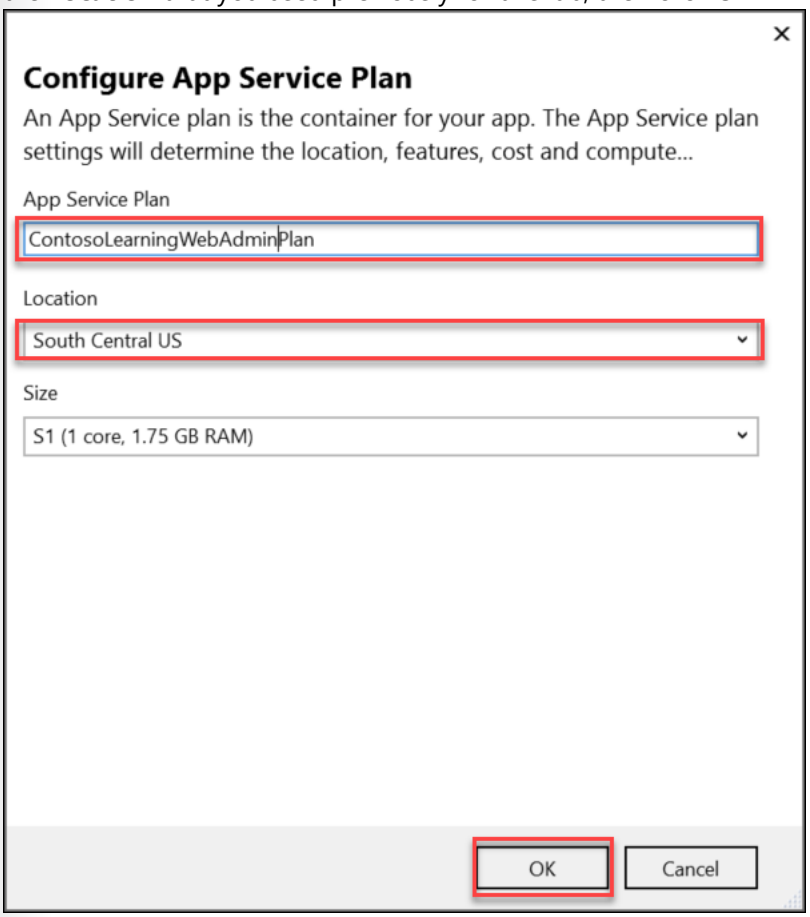
2. Click on **Microsoft Azure App Service**, then select the **Create New** radio button, then click **Publish**



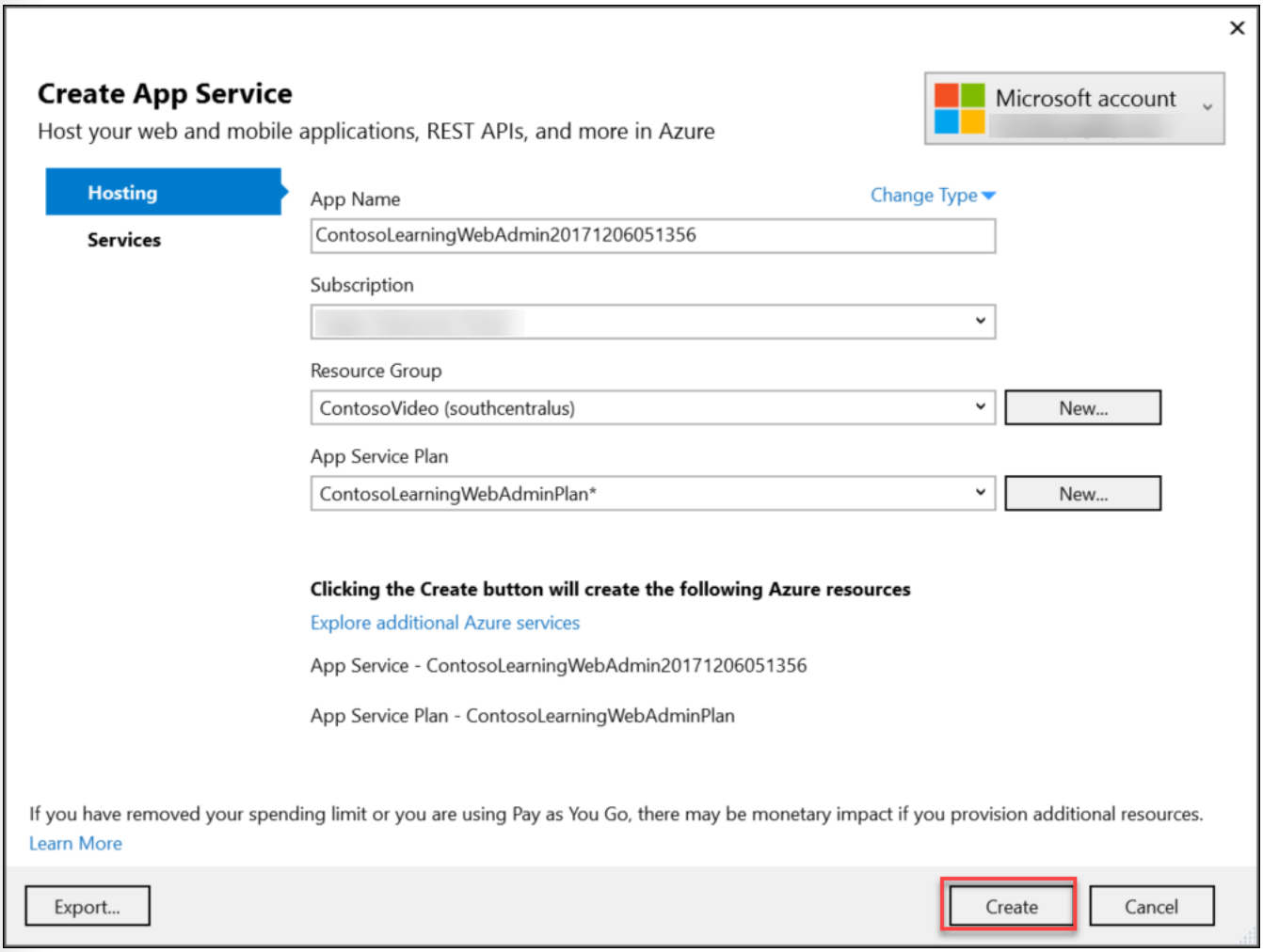

3. On the **Create App Service** dialog, select the **ContosoVideo** Resource Group, then click **New** next to **App Service Plan**



4. On the **Configure App Service Plan** dialog, enter a valid name into the **App Service Plan** name field, then select the **Location** that you used previously for this lab, then click **OK**

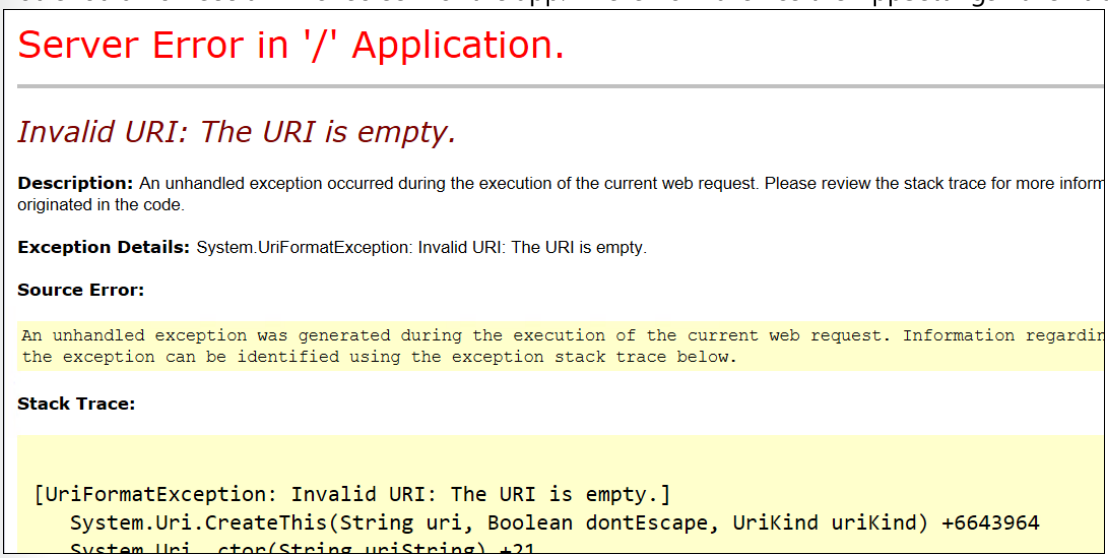


5. Click **Create**



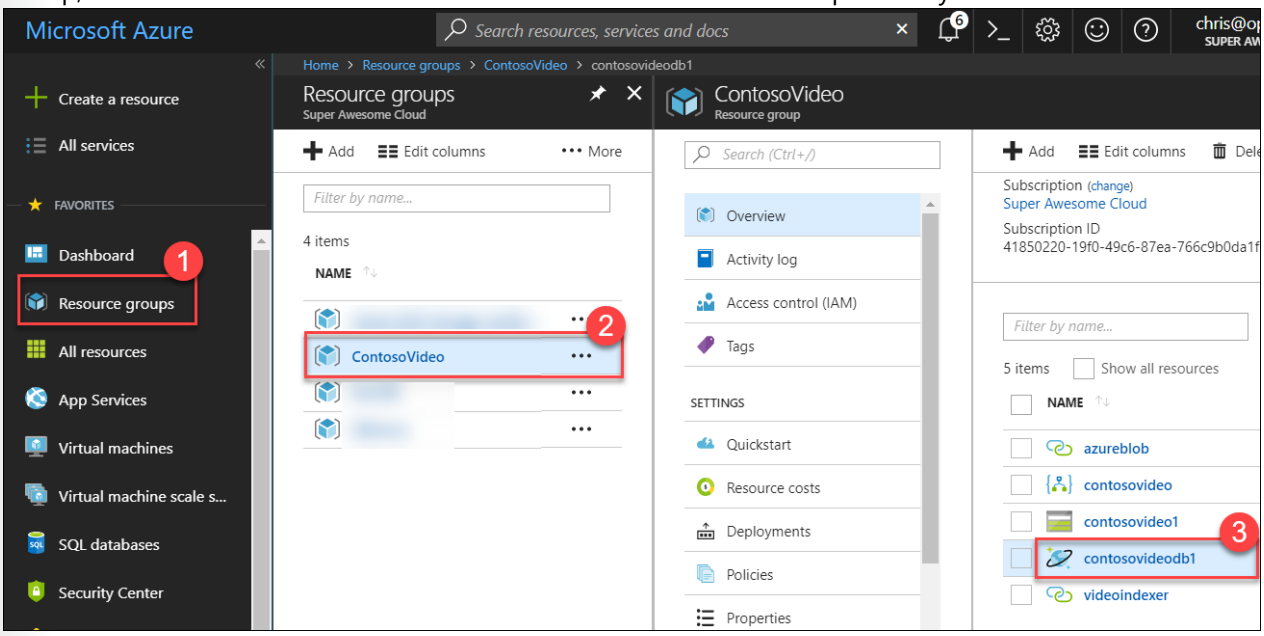

6. Once the application has finished deploying to the Azure Web App, a new browser window will be opened and navigated to the web app running in Azure

7. You should now see an Error screen for the app. This is normal since the AppSettings haven't been configured yet.

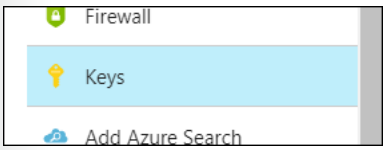



## Task 6: Configure Application Settings

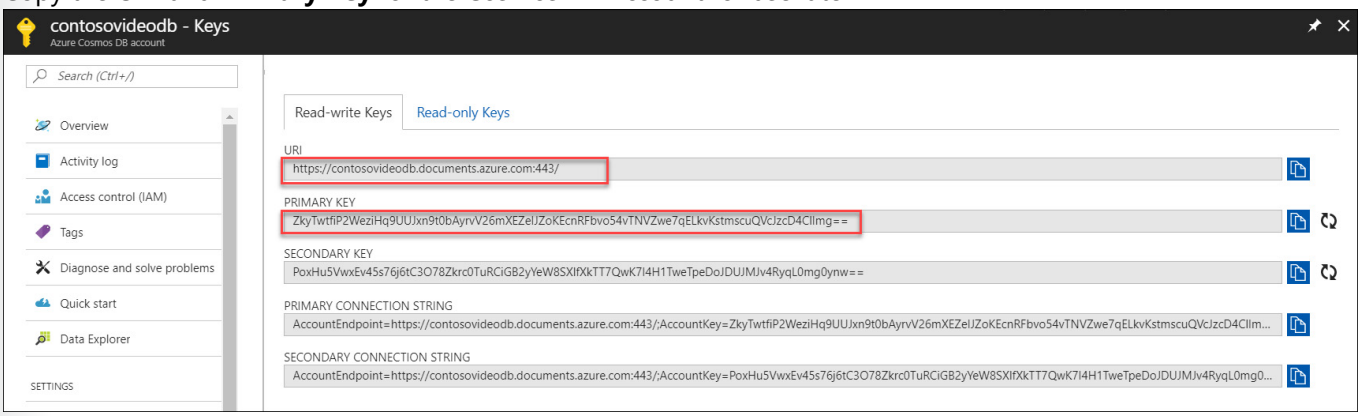
1. Open the Azure Portal, click on **Resource groups** in the left-side menu, then click the **ContosoVideo** Resource Group, then click on the **Azure Cosmos DB Account** that was created previously



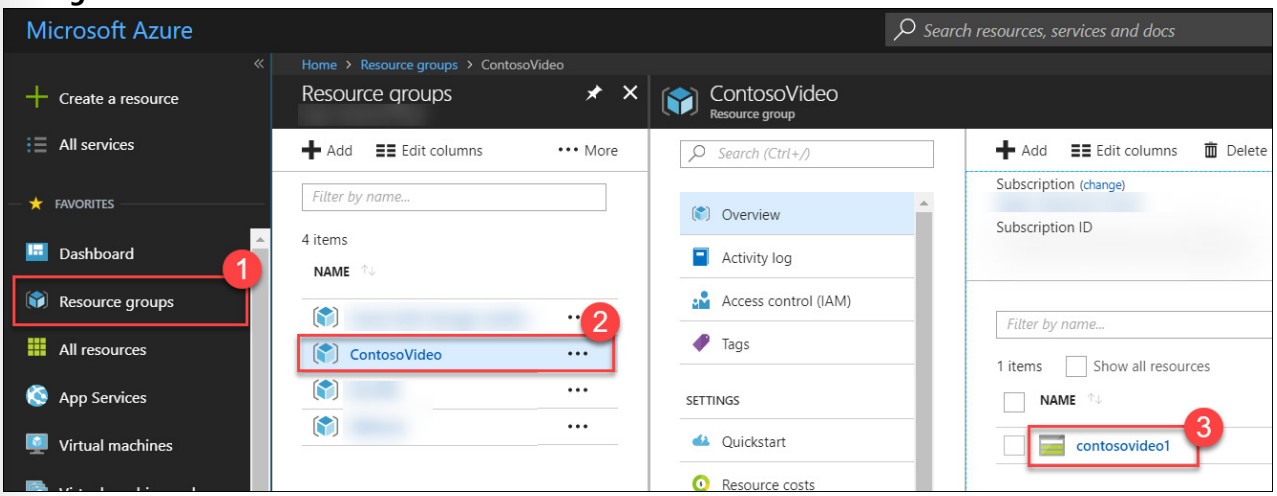

2. On the **Azure Cosmos DB Account** blade, click on **Keys**



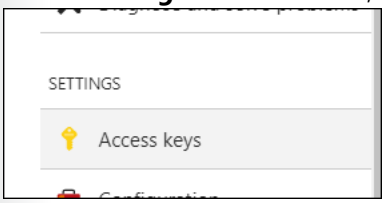
3. Copy the **URI** and **Primary Key** for the Cosmos DB Account for use later



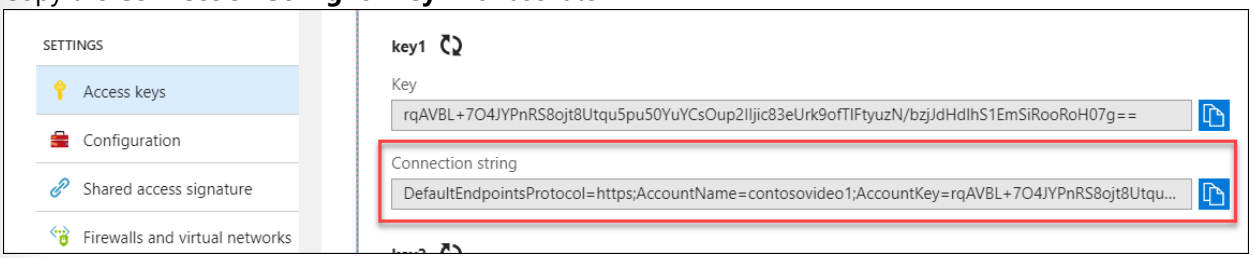

4. Click on **Resource groups** in the left-side menu, then click the **ContosoVideo** Resource Group, then click on the **Storage Account** created in Exercise 2



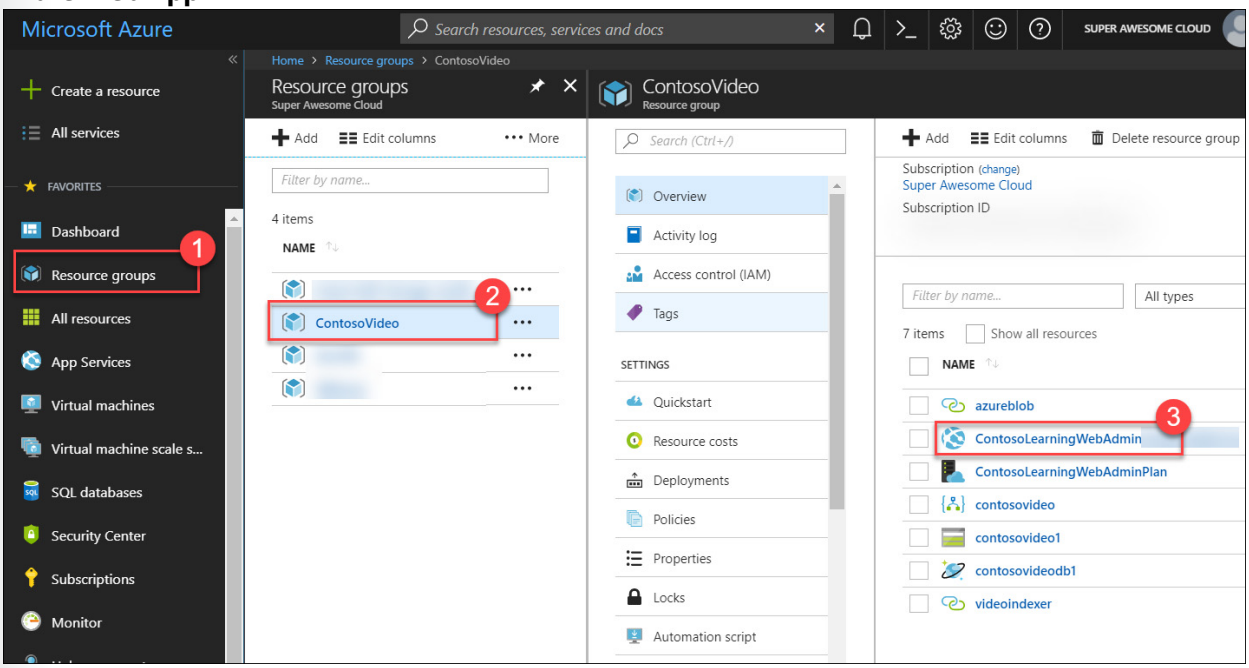

5. On the **Storage account** blade, click on **Access keys**



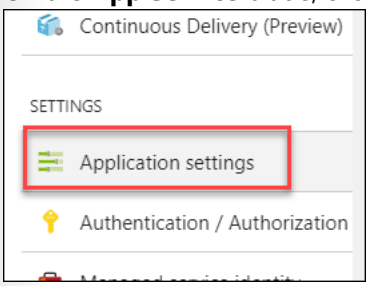

6. Copy the **Connection String** for **Key 1** for use later



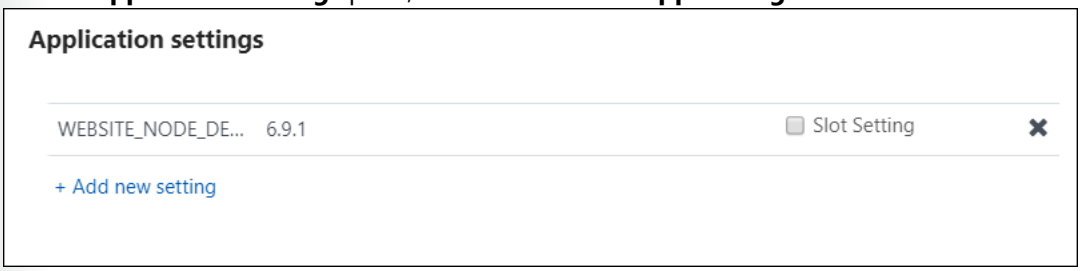
7. Click on **Resource groups** in the left-side menu, then click the **ContosoVideo** Resource Group, then click on the **Azure Web App**



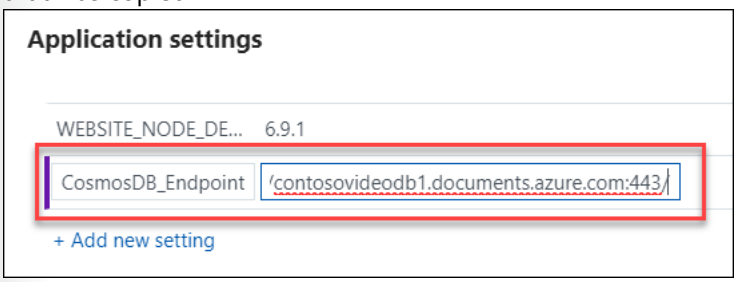

8. On the **App Service** blade, click on **Application settings**



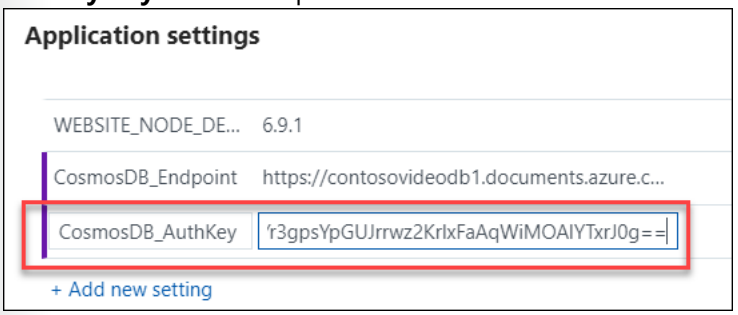

9. On the **Application settings** pane, scroll down to the **App settings** section



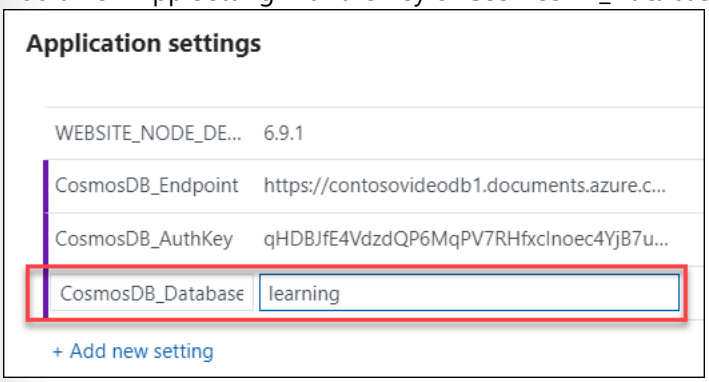
10. Add a new App Setting with the Key of **CosmosDB_Endpoint** with the Value set to the **Cosmos DB Account URI** that was copied



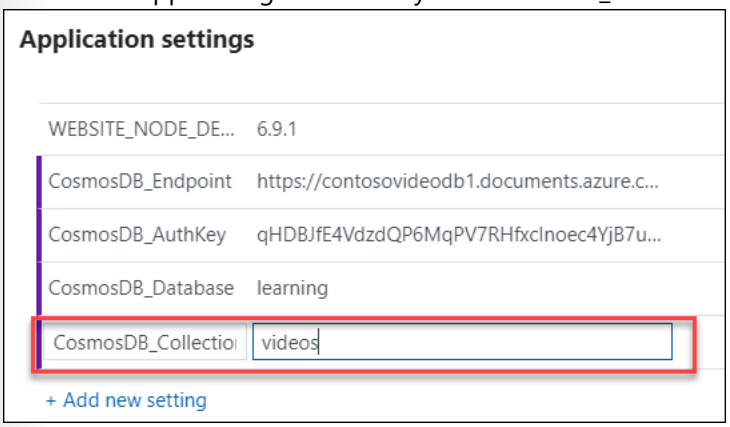

11. Add a new App Setting with the Key of **CosmosDB_AuthKey** with the Value set to the **Cosmos DB Account Primary Key** that was copied




12. Add a new App Setting with the Key of **CosmosDB_Database** with the Value set to **learning**

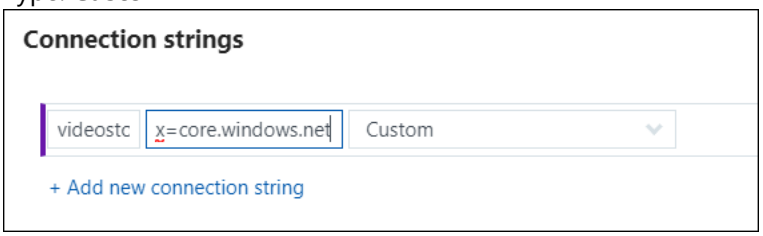



13. Add a new App Setting with the Key of **CosmosDB_Collection** with the Value set to **videos**

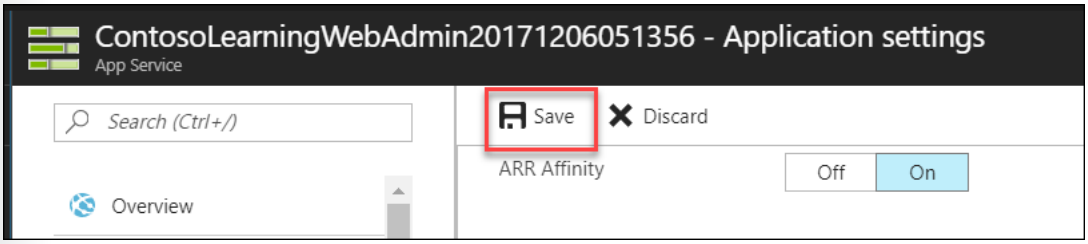


14. Add a new App Setting with the Key of **VideoIndexerAPI_Key** with the Value set to the **Video Indexer API Key** that was copied previously

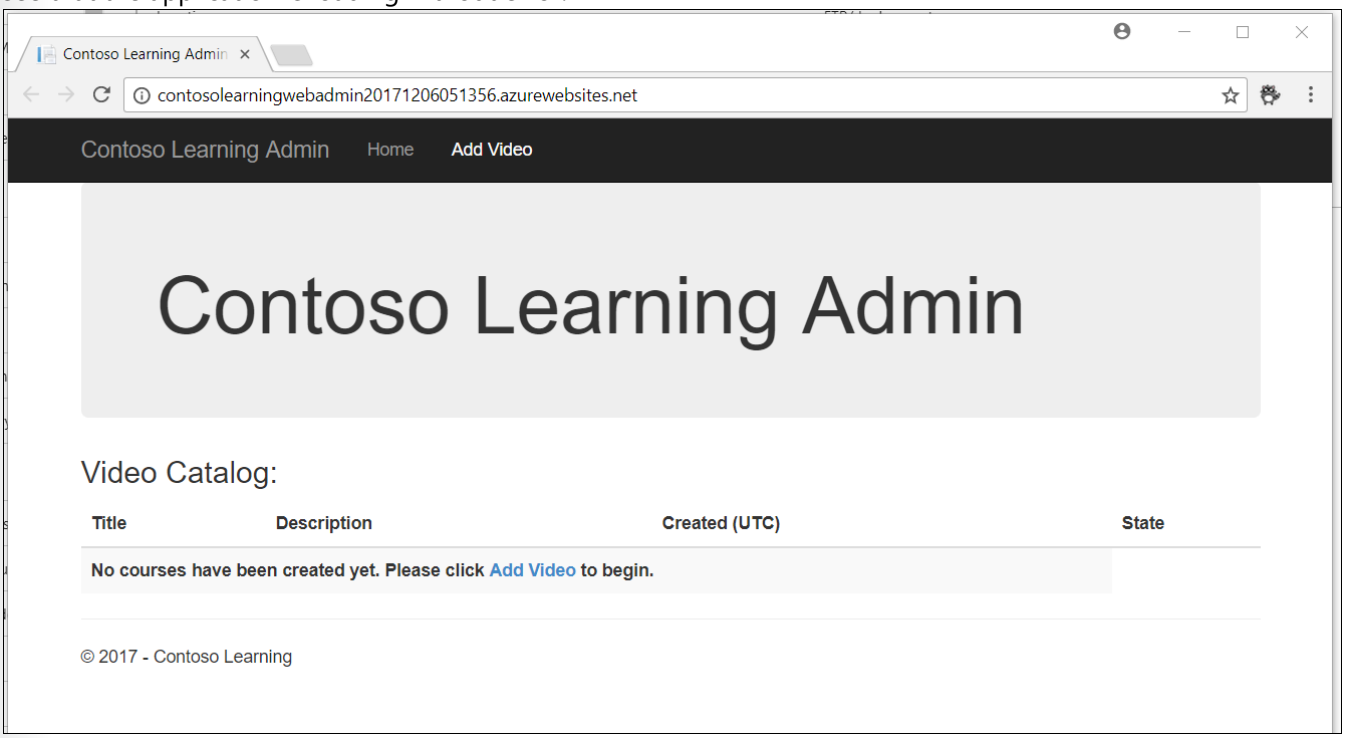

15. Locate the **Connection strings** section, and add a new Connection String with the following values:
    a. Name: **videostorage**
    b. Value: ***paste in the Storage Account connection string copied previously***
    c. Type: **Custom**




16. Click **Save**



17. Refresh the browser with the Admin Web App running in it. If you closed it, then open it up again. You will now see that the application is loading without error.




18. **Don't upload a video through the Admin app yet.** We still need to finish setting up the back-end of the application.

# Exercise 4: Update video status when processing is complete

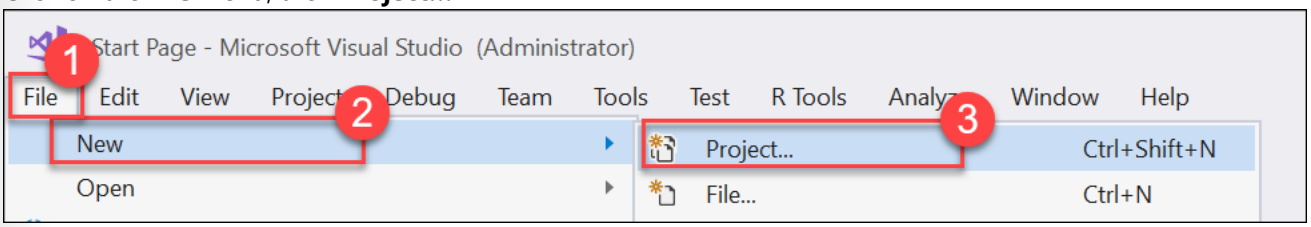
Duration: 20 minutes

In this exercise, you will integrate an Azure Function with the Logic App Workflow so that the Azure Cosmos DB database is updated when a video is finished being processed within Video Indexer.

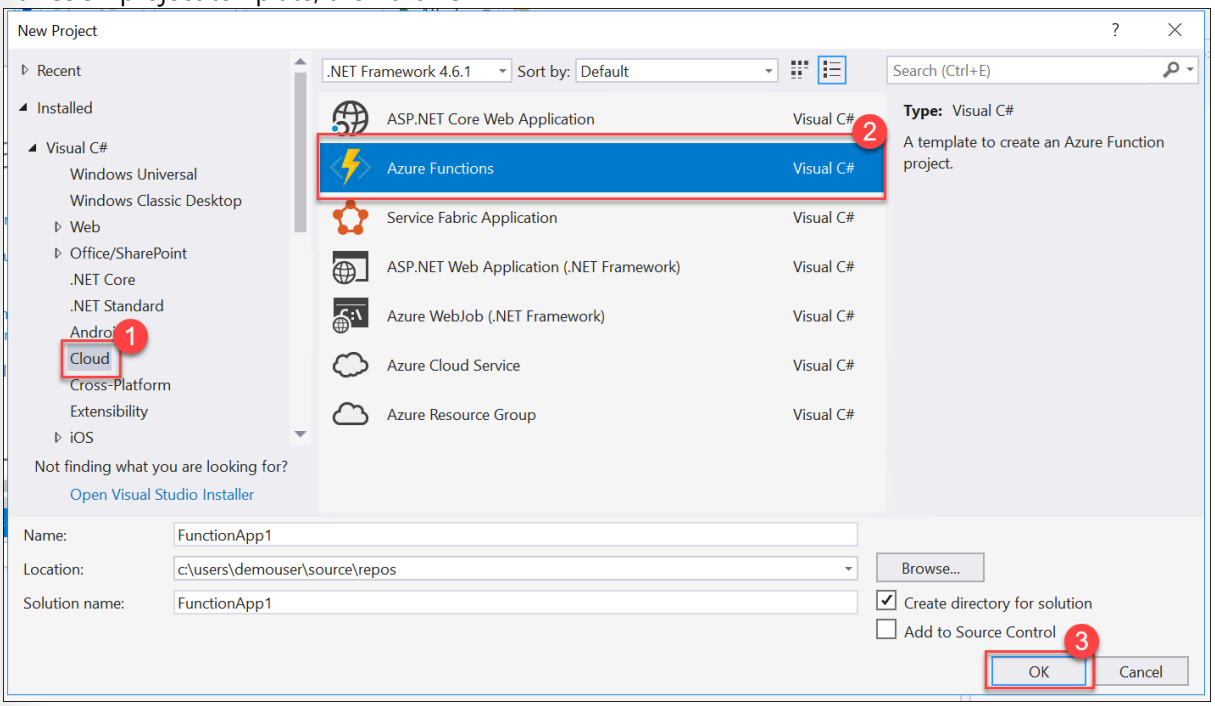
## Step 1: Create Azure Function

1. Open **Visual Studio 2017**

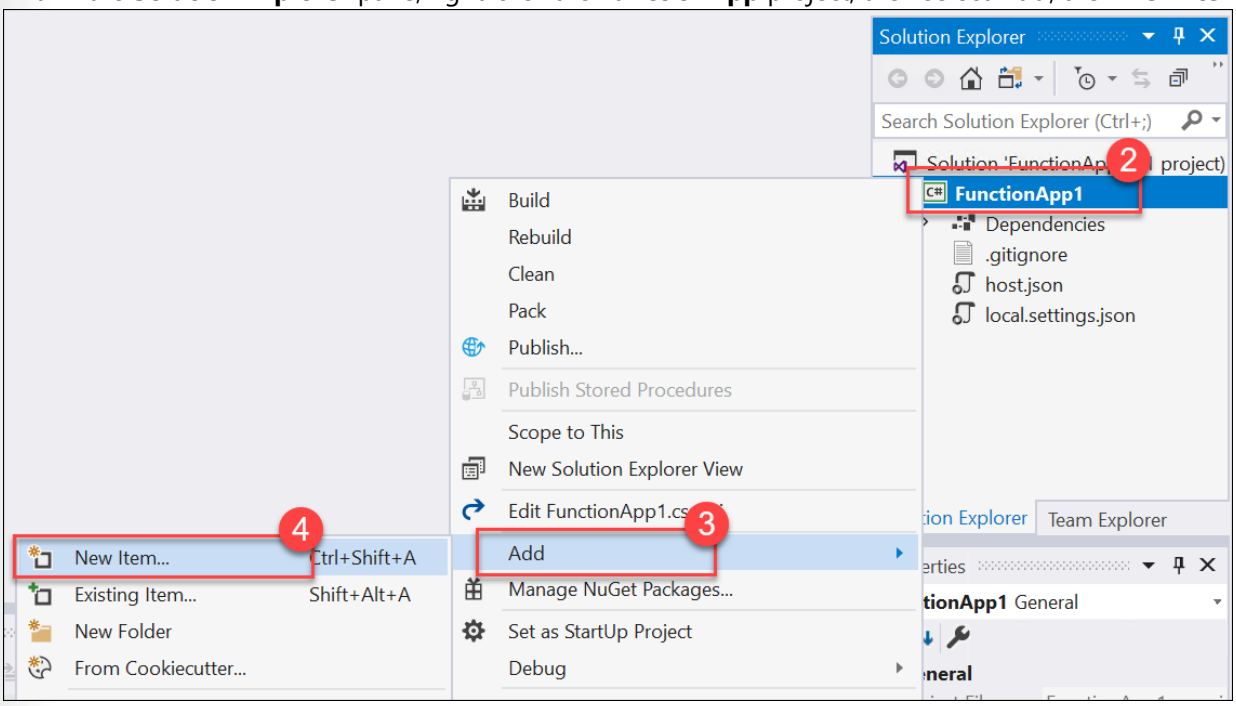
2. Click on the **File** menu, then **Project...**



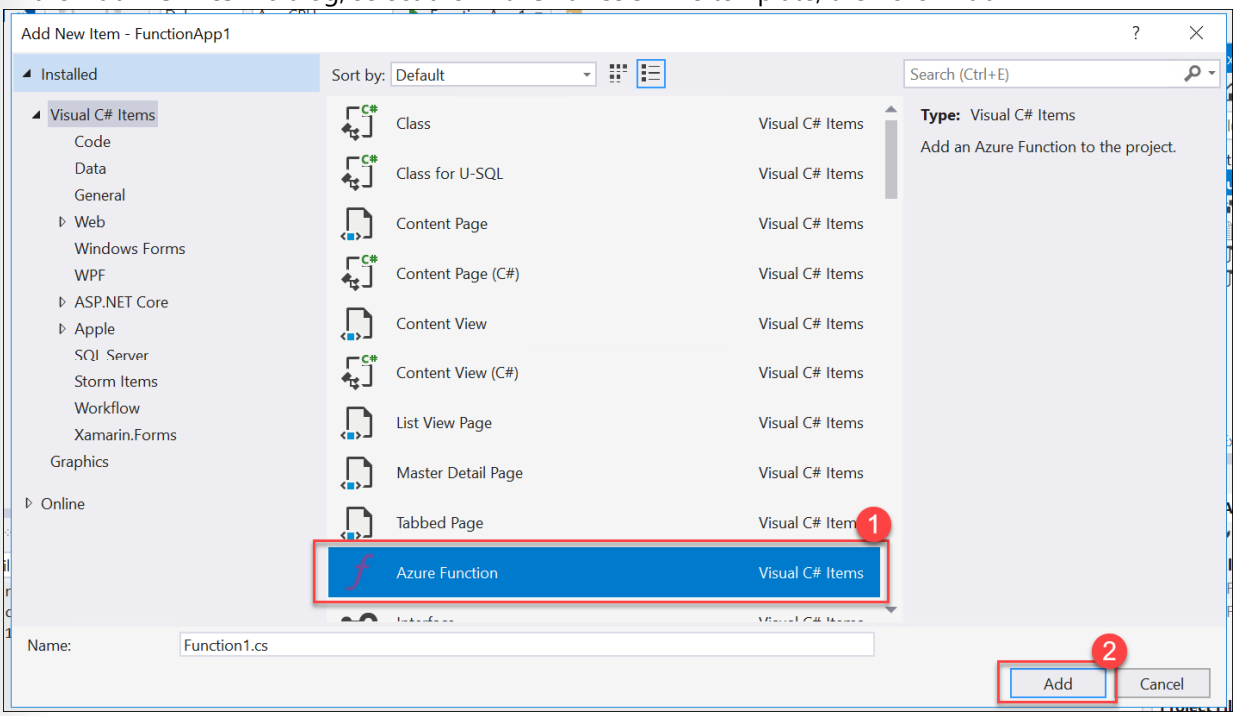

3. On the **New Project** dialog, select the **Cloud** category underneath **Visual C#** on the left, then select the **Azure Function** project template, then click **OK**



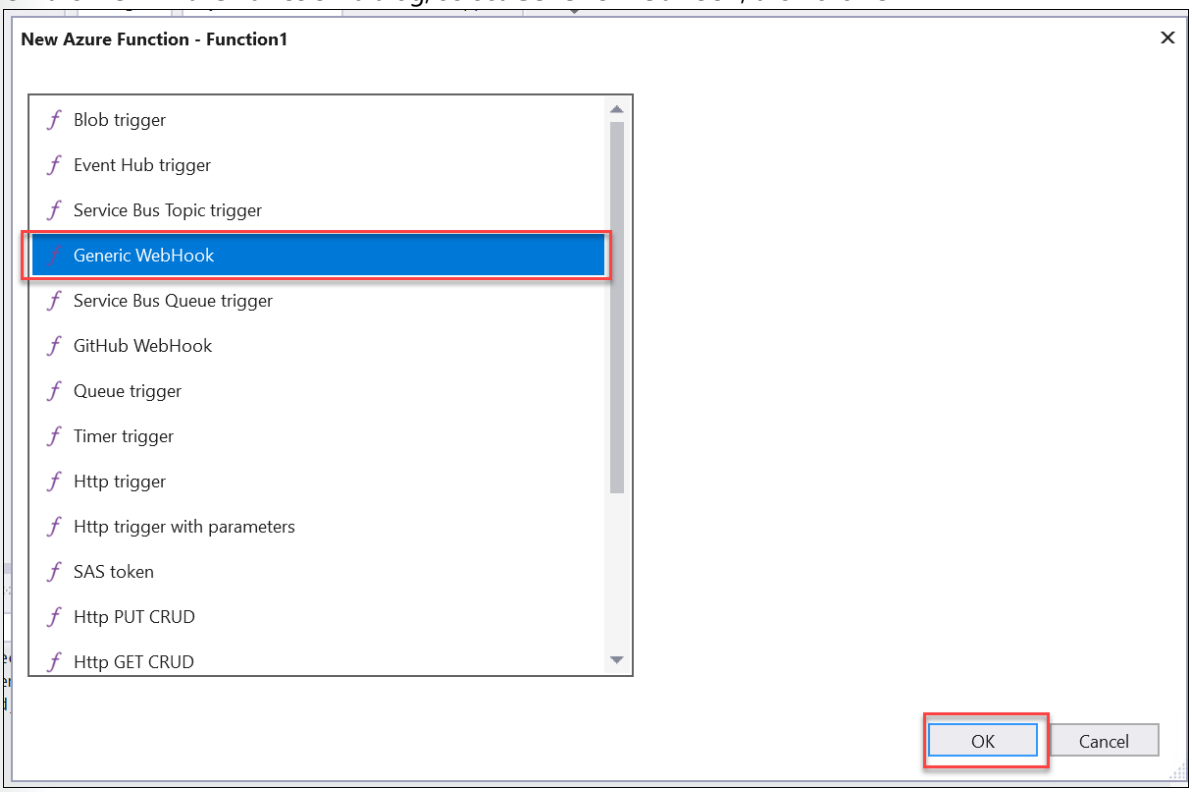
4. Within the **Solution Explorer** pane, right-click the **FunctionApp** project, then select **Add**, then **New Item...**



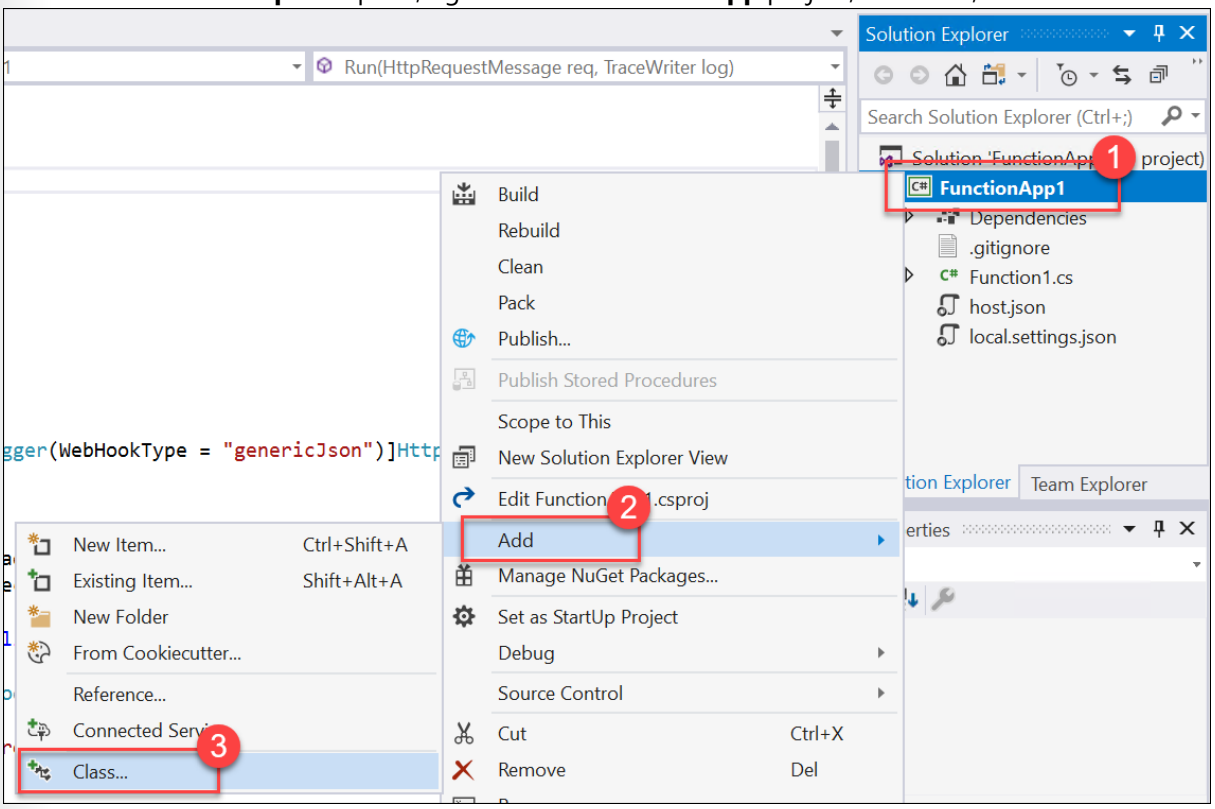

5. In the **Add New Item** dialog, select the **Azure Function** file template, then click **Add**



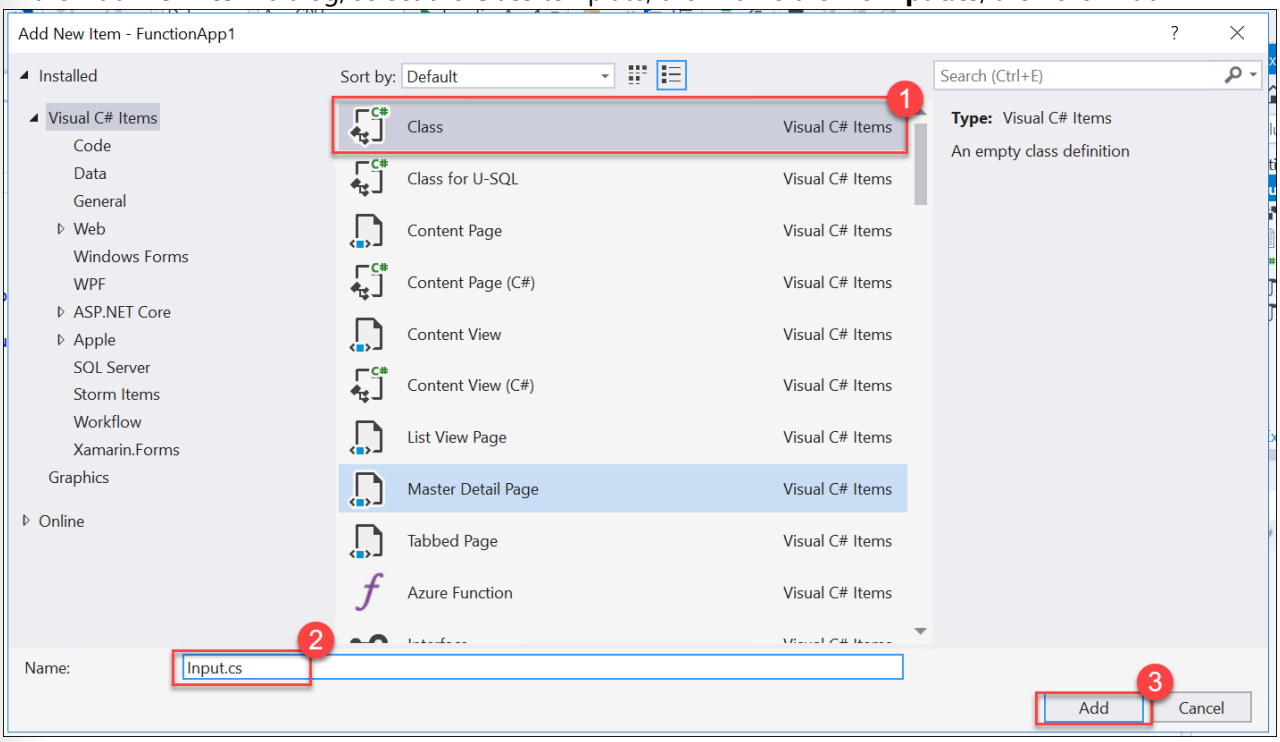
6. On the **New Azure Function** dialog, select **Generic WebHook**, then click **OK**




7. Within the **Solution Explorer** pane, right-click the **FunctionApp** project, click **Add**, then **Class...**



8. In the **Add New Item** dialog, select the **Class** template, then name the file **Input.cs**, then click **Add**




9. In the code for the **Input.cs** class, update it to match the below **Input** class with the following two properties

```
public class Input
{
    public string documentId { get; set; }
    public string videoId { get; set; }
}
```

10. Save the file

11. Open the **Function1.cs** file for the new Function that was previously created

12. Update signature of the **Run** method to contain an **input** parameter of type **Input**. This will allow the Azure Function to automatically pull in the parameter values (documentId, videoId) that are passed in through the Request Body and parse them into this Input parameter value.

```
public static async Task<object> Run(
    [HttpTrigger]Input input,
    HttpRequestMessage req,
    TraceWriter log)
```

13. Replace the body of the **Run** method with the following code that checks the necessary parameters are passed in to the method

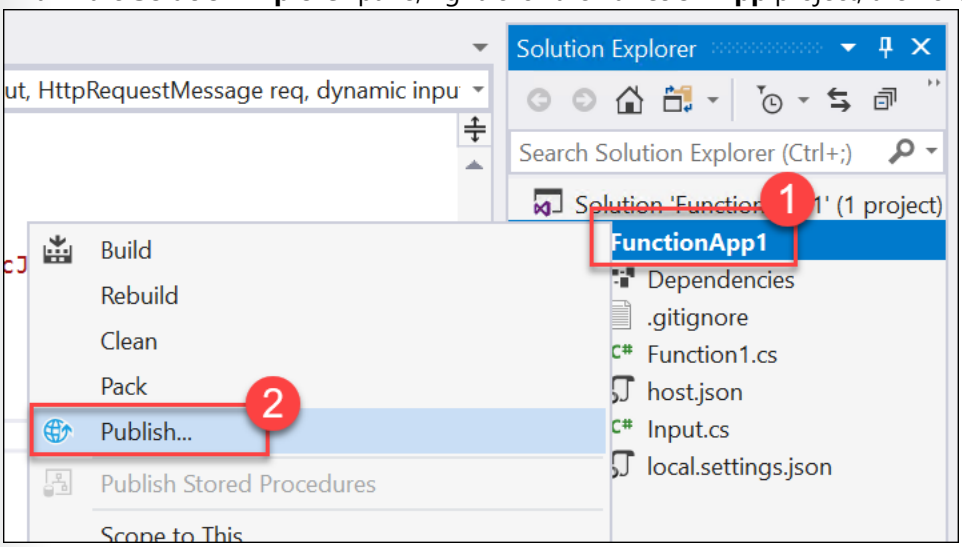
```
log.Info("Function triggered...");

// Log the parameters passed in through the Request Body
log.Info($"DocumentId: {input.documentId}");
log.Info($"VideoId: {input.videoId}");

if (string.IsNullOrEmpty(input.videoId) || string.IsNullOrEmpty(input.documentId))
{
    log.Error("DocumentId and/or VideoId parameter missing!");
    return req.CreateResponse(HttpStatusCode.BadRequest, $"Please pass a 'videoId' and 'documentId'
in the Http request body");
}

return req.CreateResponse(HttpStatusCode.OK, "Success");
```

14. The resulting **Run** method should look as follows:

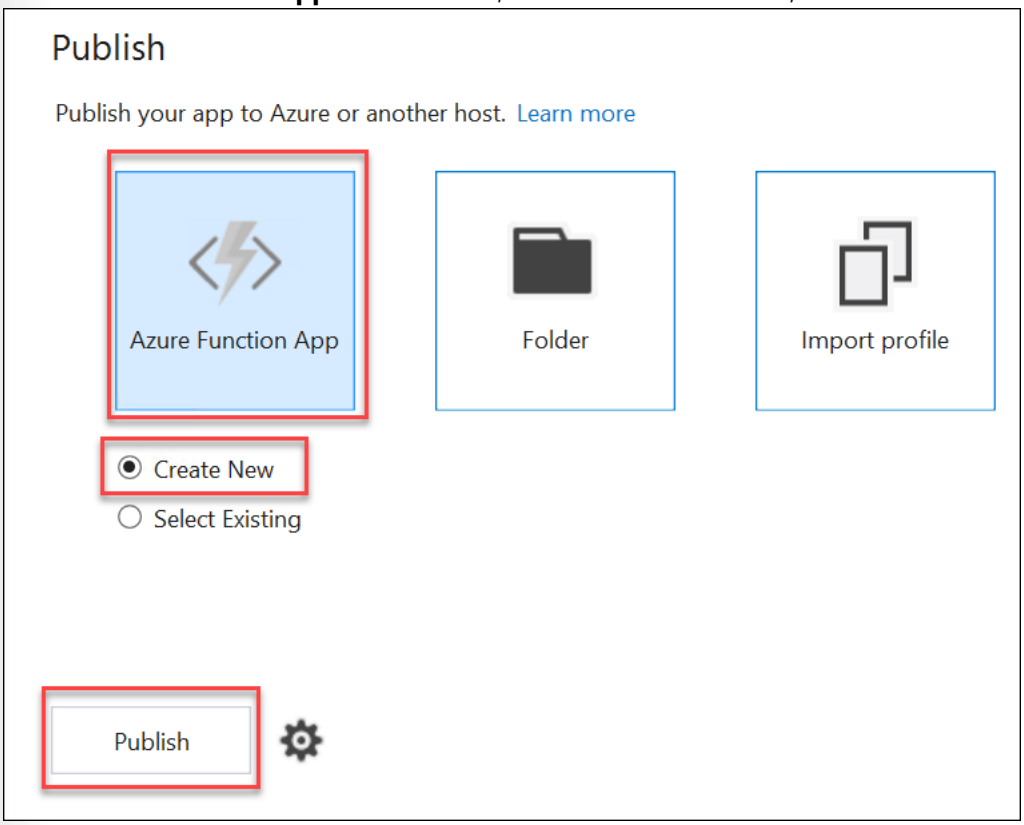
```
[FunctionName("Function1")]
public static async Task<object> Run(
    [HttpTrigger]Input input,
    HttpRequestMessage req,
    TraceWriter log)
{
    log.Info("Function triggered...");

    // Log the parameters passed in through the Request Body
    log.Info($"DocumentId: {input.documentId}");
    log.Info($"VideoId: {input.videoId}");

    if (string.IsNullOrEmpty(input.videoId) || string.IsNullOrEmpty(input.documentId))
    {
        log.Error("DocumentId and/or VideoId parameter missing!");
        return req.CreateResponse(HttpStatusCode.BadRequest, $"Please pass a 'videoId' and
'documentId' in the Http request body");
    }

    return req.CreateResponse(HttpStatusCode.OK, "Success");
}
```

15. Save the source code files

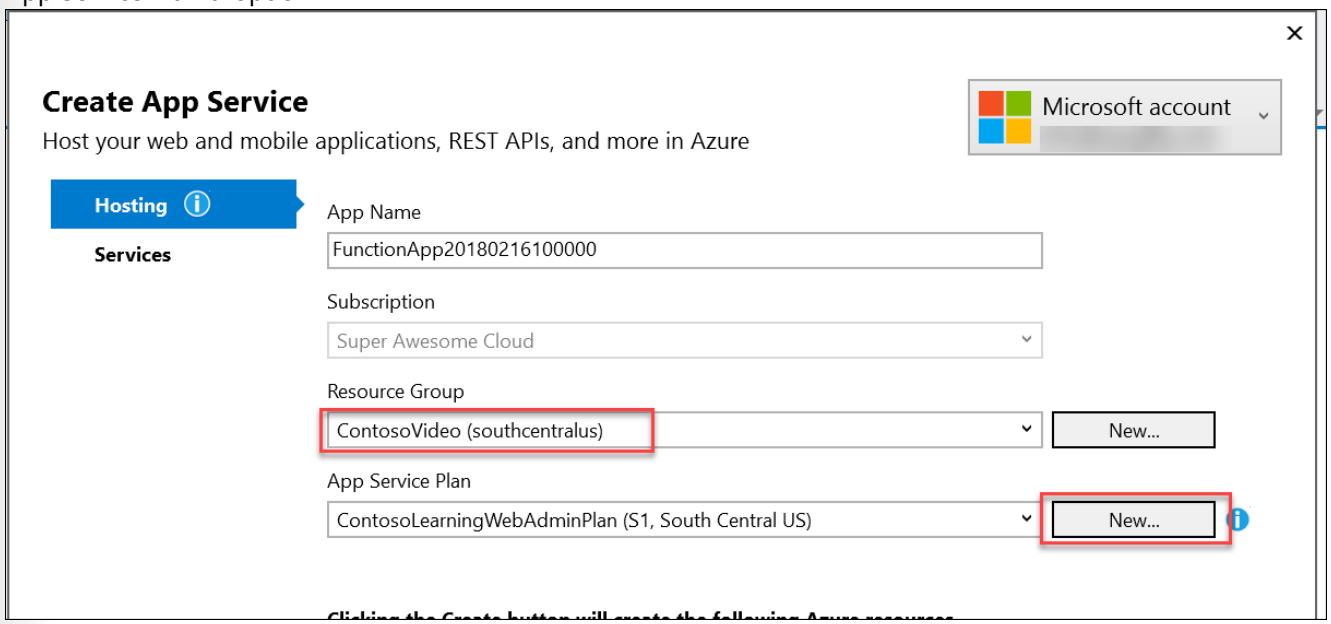
16. Within the **Solution Explorer** pane, right-click the **Function App** project, then click **Publish**



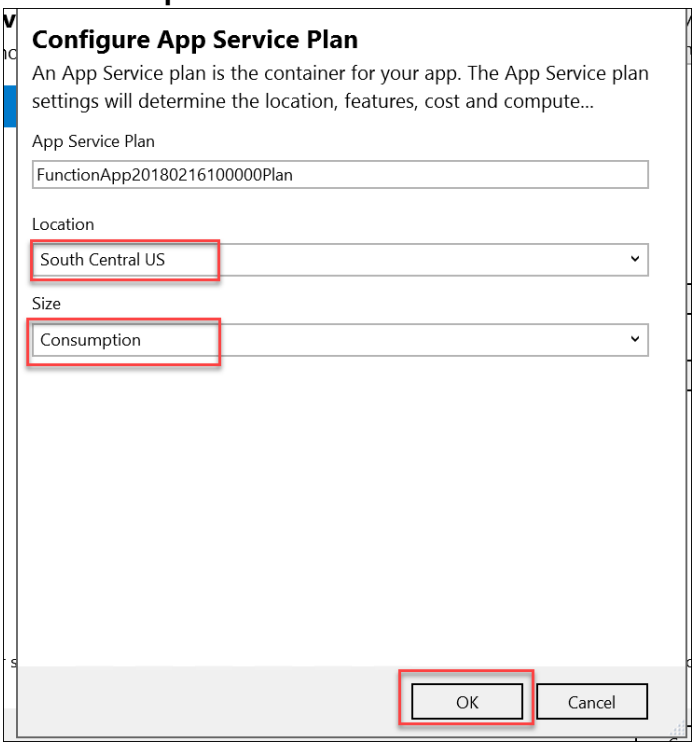

17. Select **Azure Function App** under Publish, then select **Create New**, then click **Publish**



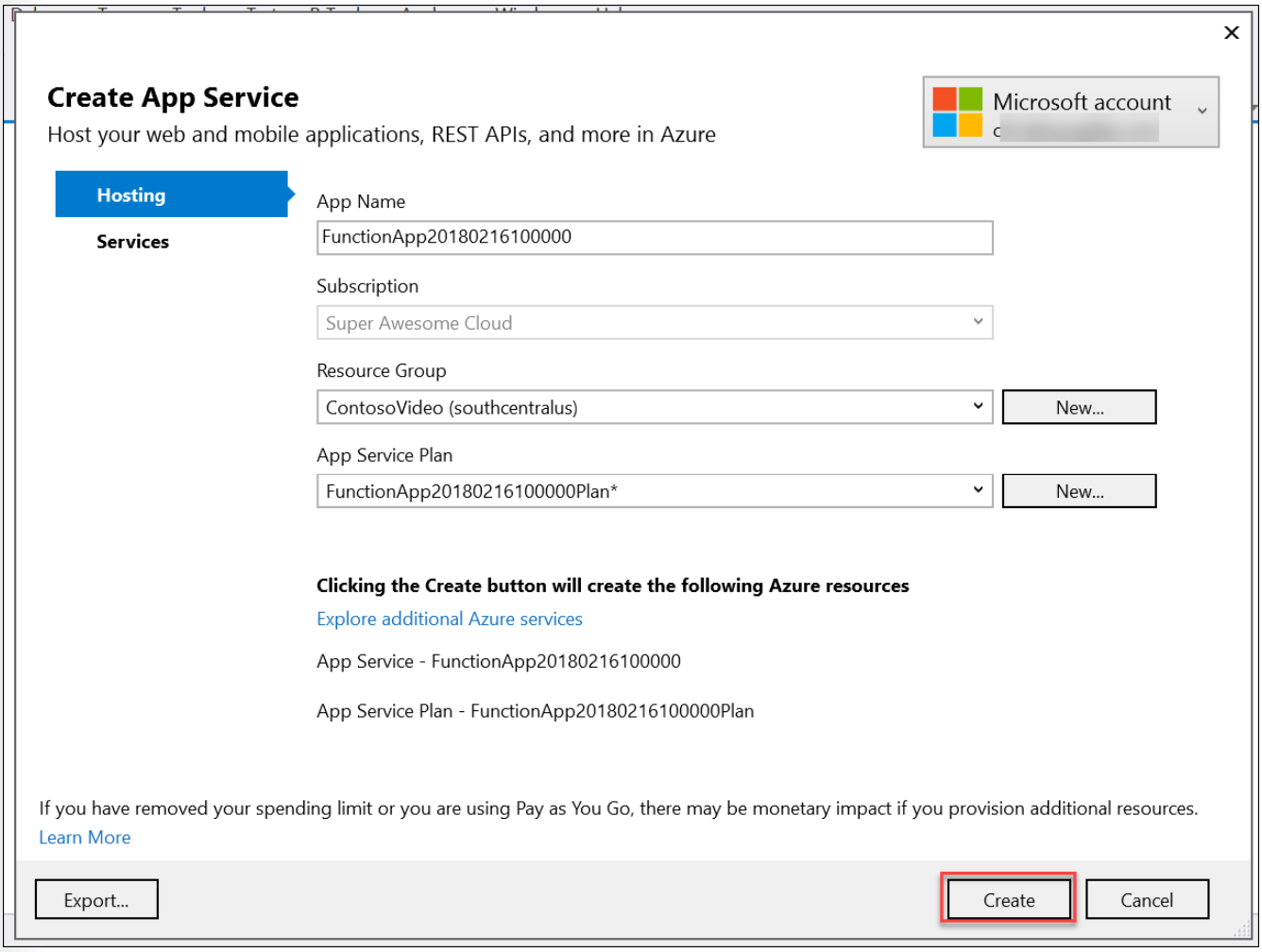
18. On the **Create App Service** dialog, select the **ContosoVideo** Resource Group, then click on **New...** next to the App Service Plan dropdown



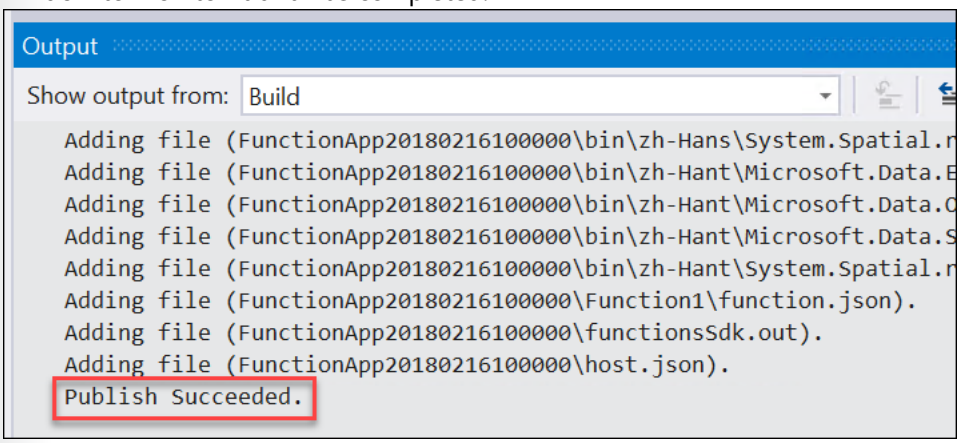

19. On the Configure App Service Plan dialog, enter the following values, then click **OK**:
    a. Location: **choose the same location you used for the rest of the lab**
    b. Size: **Consumption**



20. Click **Create**

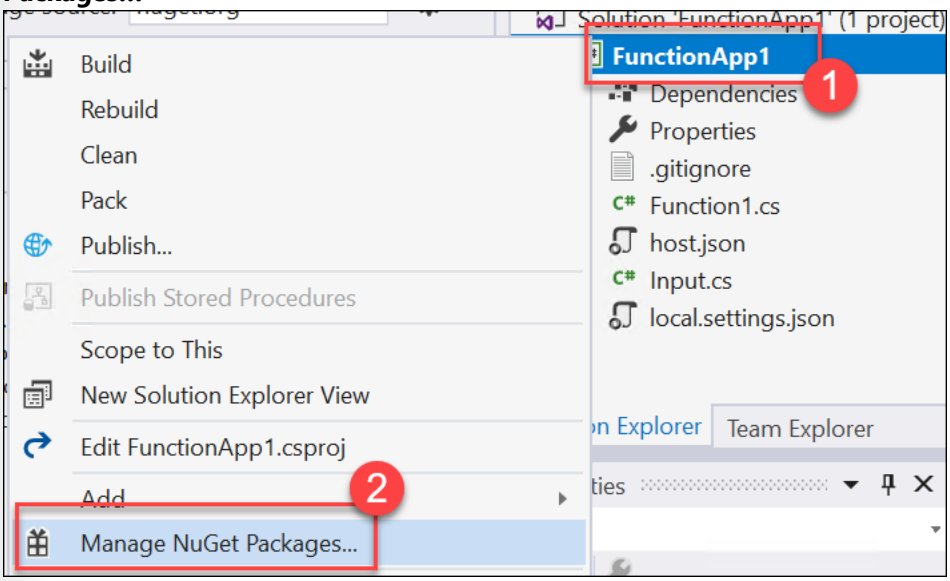



21. Wait for the publish to complete. This should take about 1 minute to complete. You can watch the **Output** window to monitor it until it's completed.

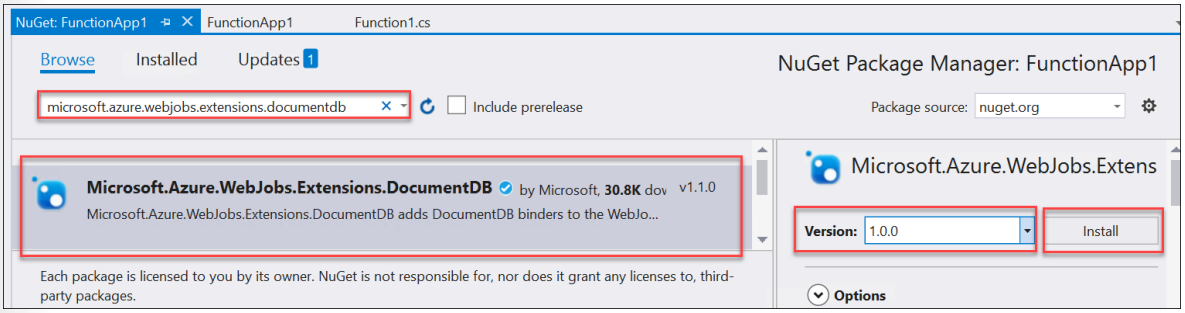


## Step 2: Update Cosmos DB Document with Video Processing State

1. With the **Function App** still open in Visual Studio, right-click on the **Function App**, then click **Manage NuGet Packages...**




2. Search for, and install the **Microsoft.Azure.WebJobs.Extensions.DocumentDB** NuGet package. Be sure to select **Version 1.0.0** to install. Do NOT choose the latest version, as it will not install due to a version conflict.




3. Open the **Function1.cs** file for the Azure Function code

4. Modify the **Run** method declaration to include the following "*dynamic inputDocument*" parameter and **DocumentDB** attribute

```
[DocumentDB("learning", "videos", Id = "{documentId}", ConnectionStringSetting =
"contosovideodb_DOCUMENTDB")] dynamic inputDocument,
```

The first 2 parameters of the **DocumentDB** attribute define to connect to the "*videos*" Cosmos DB Collection within the "*learning*" database. And the value of "*{documentId}*" will enable it to retrieve the Document whose ID is set to the same value of the "*documentId*" value passed into the method via the HTTP POST. The "*ConnectionStringSetting*" parameter sets the name of the App Setting that will store the Cosmos DB Connection String.

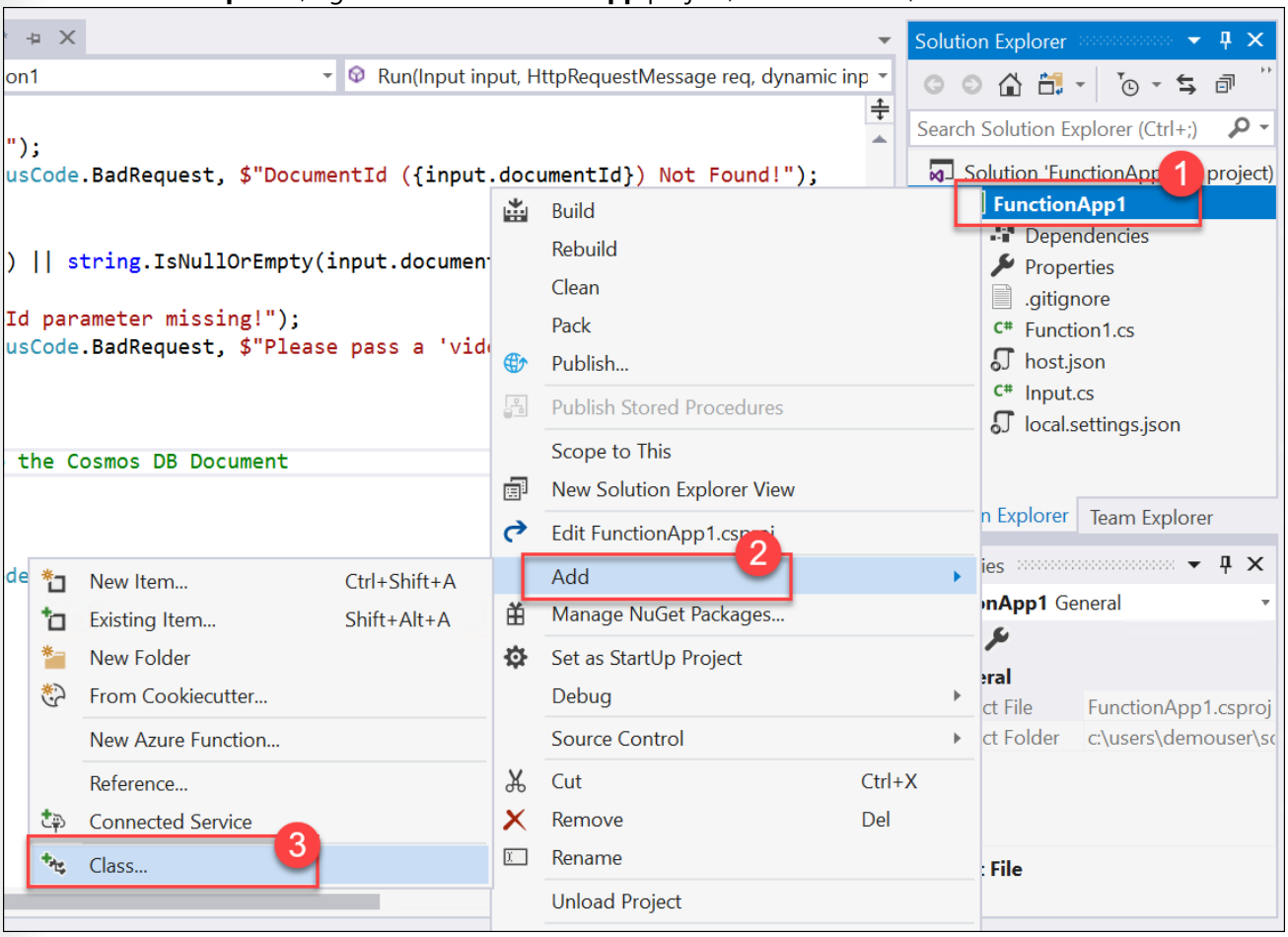
5. Now, the full **Run** method signature should look as follows:

```
public static async Task<object> Run(
    [HttpTrigger]Input input,
    HttpRequestMessage req,
    [DocumentDB("learning", "videos", Id = "{documentId}", ConnectionStringSetting =
"contosovideodb_DOCUMENTDB")] dynamic inputDocument,
    TraceWriter log)
```

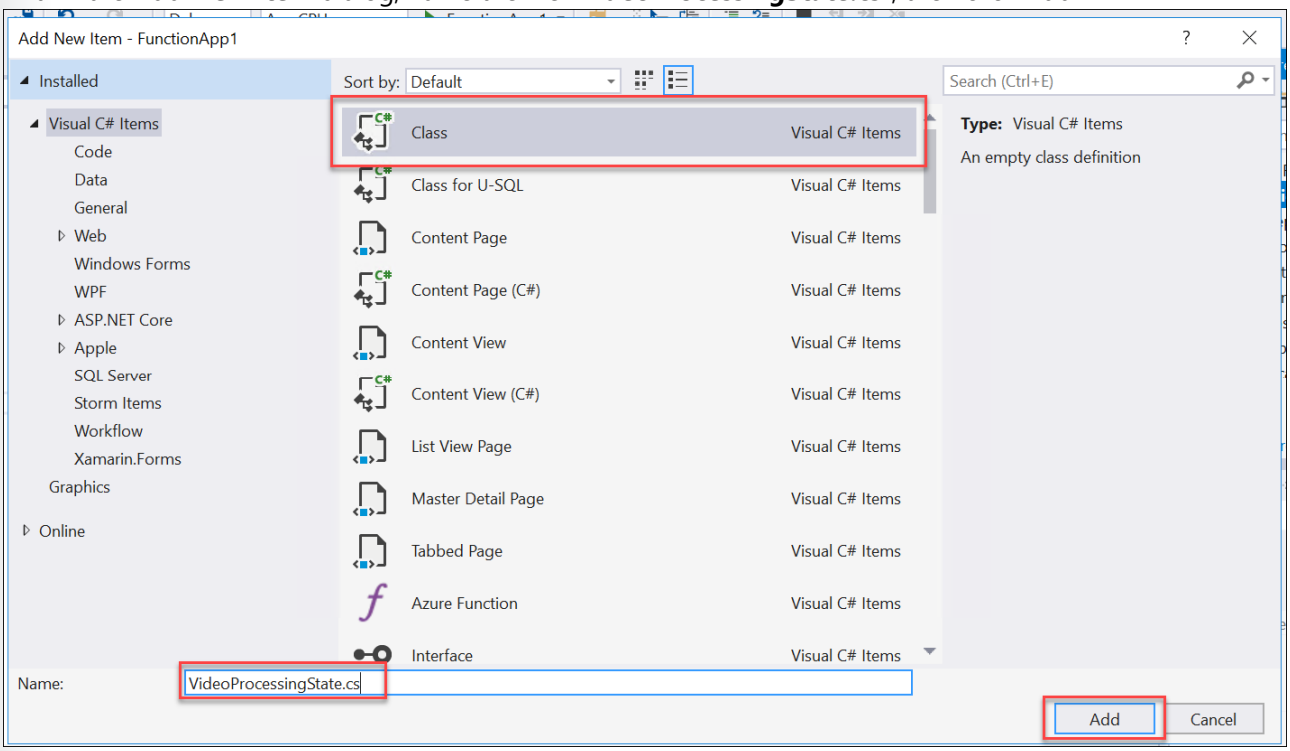
6. Immediately before the fine **return** statement of the **Run** method, insert the following line of code that will update the **videoId** property of the Cosmos DB Document to contain the videoId that is passed into the Function.

```
// Save the Video Indexer 'videoId' to the Cosmos DB Document
inputDocument.videoId = input.videoId;
```

7. Within **Solution Explorer**, right-click the **FunctionApp** project, then click **Add**, then **Class...**



8. Within the **Add New Item** dialog, name the file "**VideoProcessingState.cs**", then click **Add**




9. Replace the autogenerated **VideoProcessingState** class stub with the following code:

```
public class VideoProcessingState
{
    public string state { get; set; }
    public string progress { get; set; }

    public string ErrorType { get; set; }
    public string Message { get; set; }
}
```

10. Save the file

11. Open the **Function.cs** code file, and paste in the following **GetVideoProcessingState** method below the **Run** method:

```
private static async Task<VideoProcessingState> GetVideoProcessingState(string videoId, TraceWriter log)
{
    var client = new HttpClient();

    // Request headers
    string subscriptionKey = System.Configuration.ConfigurationManager.AppSettings["VideoIndexerAPI_Key"];
    client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", subscriptionKey);

    var uri = $"https://videobreakdown.azure-api.net/Breakdowns/Api/Partner/Breakdowns/{videoId}/State";

    var response = await client.GetAsync(uri);

    var content = await response.Content.ReadAsStringAsync();

    log.Info($"Processing State JSON: {content}");

    return JsonConvert.DeserializeObject<VideoProcessingState>(content);
}
```

12. Go back to the **Run** method, and add the following source code immediately below the "*inputDocument.videoId =*" line that will call the newly created function to load the Video Processing State

```
// Load Video Processing State
dynamic processingState = await GetVideoProcessingState(input.videoId, log);
```

13. Next, paste in the following code that checks if an Error Message was returned from the call to the Video Indexer API

```
if (!string.IsNullOrEmpty(processingState.ErrorType))
{
    log.Error($"{processingState.ErrorType}: {processingState.Message}");
    return req.CreateResponse(
        HttpStatusCode.InternalServerError,
        new { ErrorType = processingState.ErrorType, Message = processingState.Message }
    );
}
```

14. Next, paste in the following code that sets the values of the **Processing State** and **Processing Progress** to properties on the **Cosmos DB Document**; as well as logs their values to the Azure Function Logs

```
log.Info($"Video Processing State: {processingState.state}");
log.Info($"Video Processing Progress: {processingState.progress}");

// Save Video Processing State in Cosmos DB Document
inputDocument.processingState = processingState.state;
inputDocument.processingProgress = processingState.progress;
```

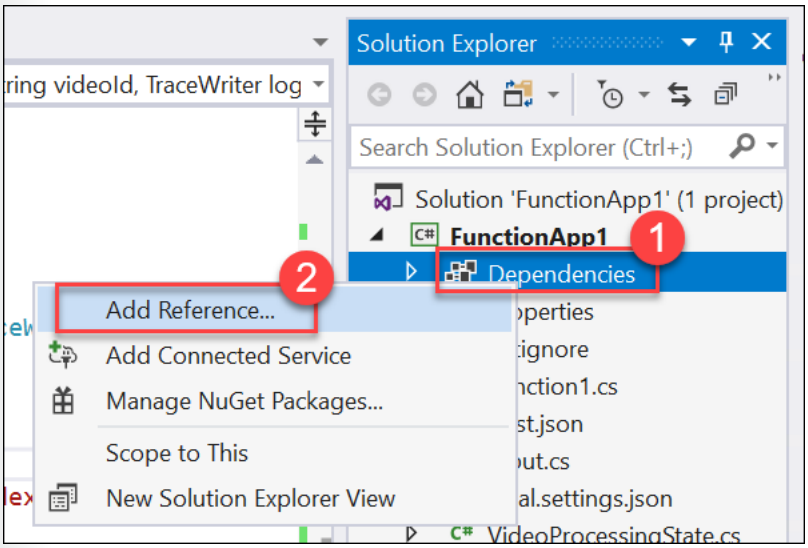
15. The final **Run** method, should look as follows:

```
[FunctionName("Function1")]
public static async Task<object> Run(
    [HttpTrigger]Input input,
    HttpRequestMessage req,
    [DocumentDB("learning", "videos", Id = "{documentId}", ConnectionStringSetting =
"contosovideodb_DOCUMENTDB")] dynamic inputDocument,
    TraceWriter log)
{
    log.Info("Function triggered...");

    // Log the parameters passed in through the Request Body
    log.Info($"DocumentId: {input.documentId}");
    log.Info($"VideoId: {input.videoId}");

    if (inputDocument == null)
    {
        log.Error("'documentId' not found!");
        return req.CreateResponse(HttpStatusCode.BadRequest, $"DocumentId ({input.documentId}) Not
Found!");
    }

    if (string.IsNullOrEmpty(input.videoId) || string.IsNullOrEmpty(input.documentId))
    {
        log.Error("DocumentId and/or VideoId parameter missing!");
        return req.CreateResponse(HttpStatusCode.BadRequest, $"Please pass a 'videoId' and
'documentId' in the Http request body");
    }



    // Save the Video Indexer 'videoId' to the Cosmos DB Document
    inputDocument.videoId = input.videoId;

    // Load Video Processing State
    dynamic processingState = await GetVideoProcessingState(input.videoId, log);

    if (!string.IsNullOrEmpty(processingState.ErrorType))
```

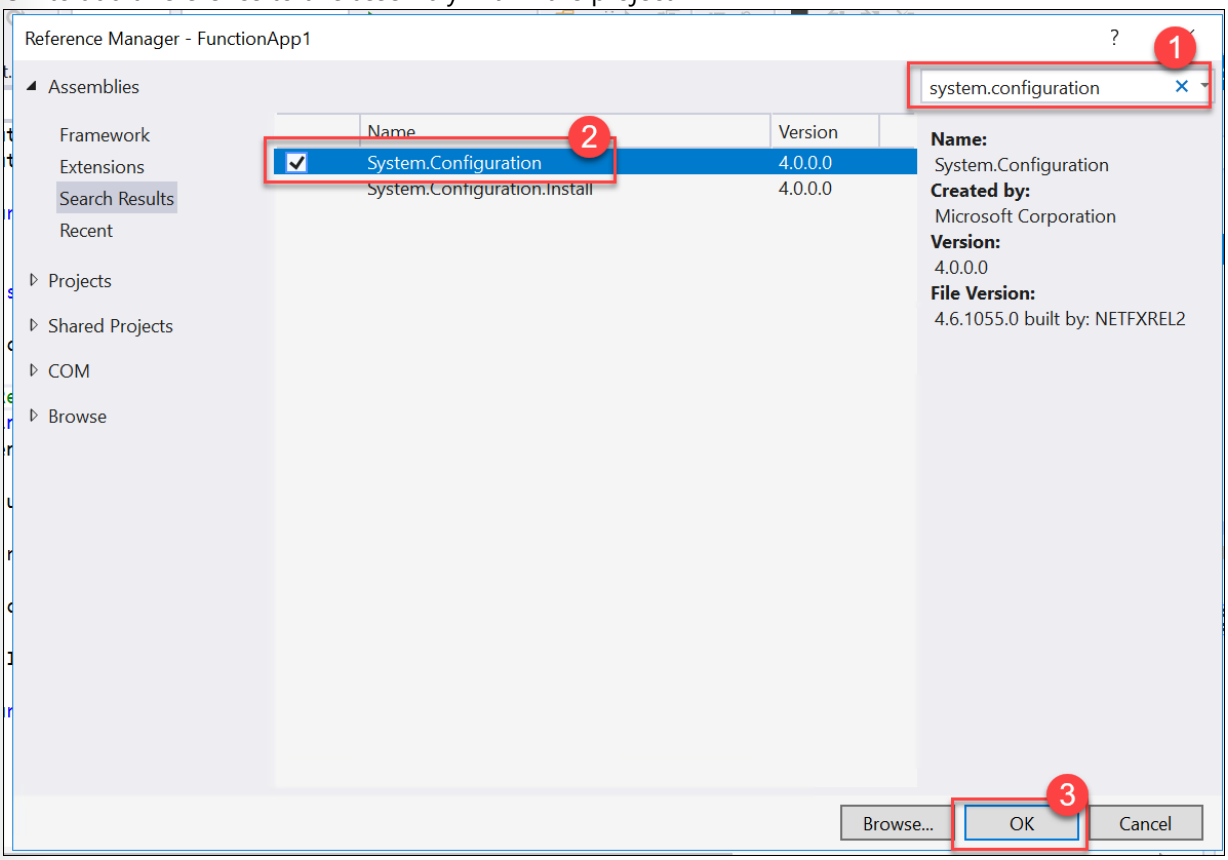
```
    {
        log.Error($"{processingState.ErrorType}: {processingState.Message}");
        return req.CreateResponse(
            HttpStatusCode.InternalServerError,
            new { ErrorType = processingState.ErrorType, Message = processingState.Message }
        );
    }

    log.Info($"Video Processing State: {processingState.state}");
    log.Info($"Video Processing Progress: {processingState.progress}");

    // Save Video Processing State in Cosmos DB Document
    inputDocument.processingState = processingState.state;
    inputDocument.processingProgress = processingState.progress;

    return req.CreateResponse(HttpStatusCode.OK, "Success");
}
```

16. Save the file

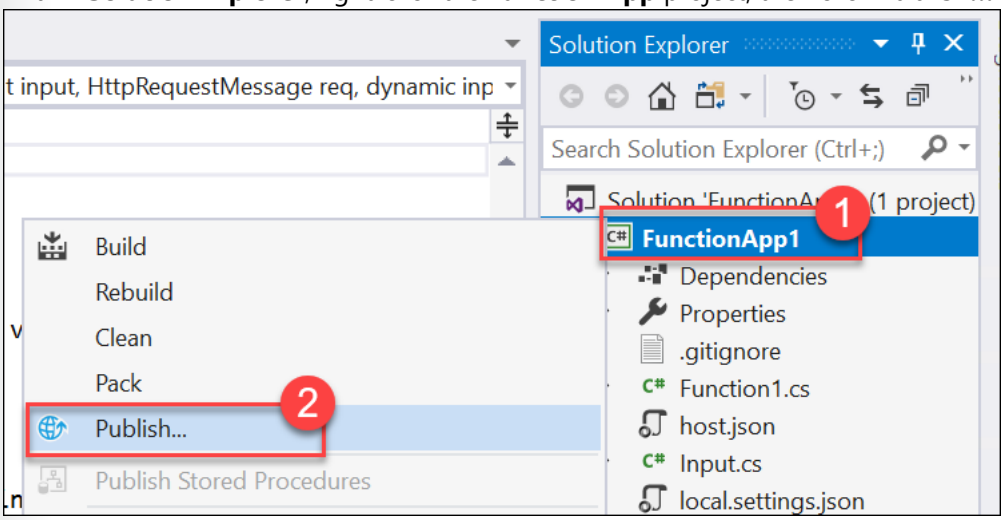
17. Within **Solution** Explorer, right-click the **Dependencies** node for the **Function App** project, then click on **Add Reference**



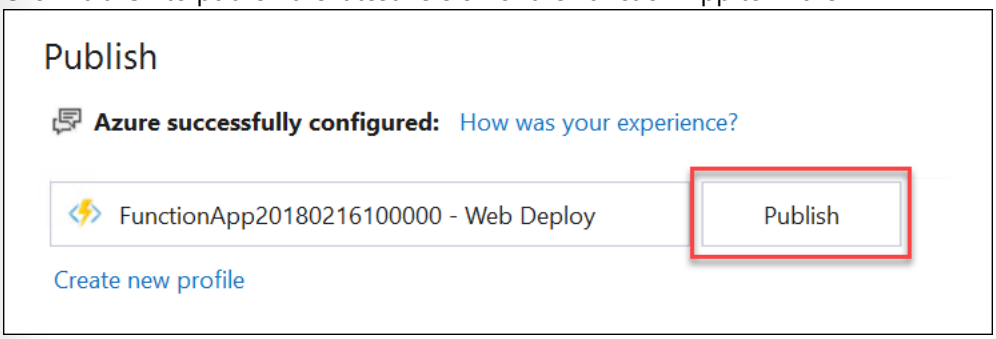
18. In the **Reference Manager** dialog, search for and locate the **System.Configuration** assembly, select it, then click **OK** to add a reference to this assembly within the project




19. Within **Solution Explorer**, right-click the **Function App** project, then click **Publish...**



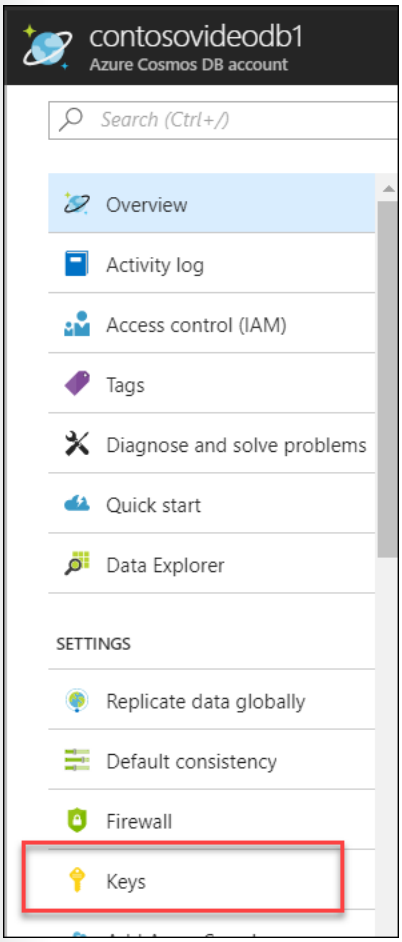
20. Click **Publish** to publish the latest version of the Function App to Azure




21. Wait for the Publish to complete, this should take about 1 minute

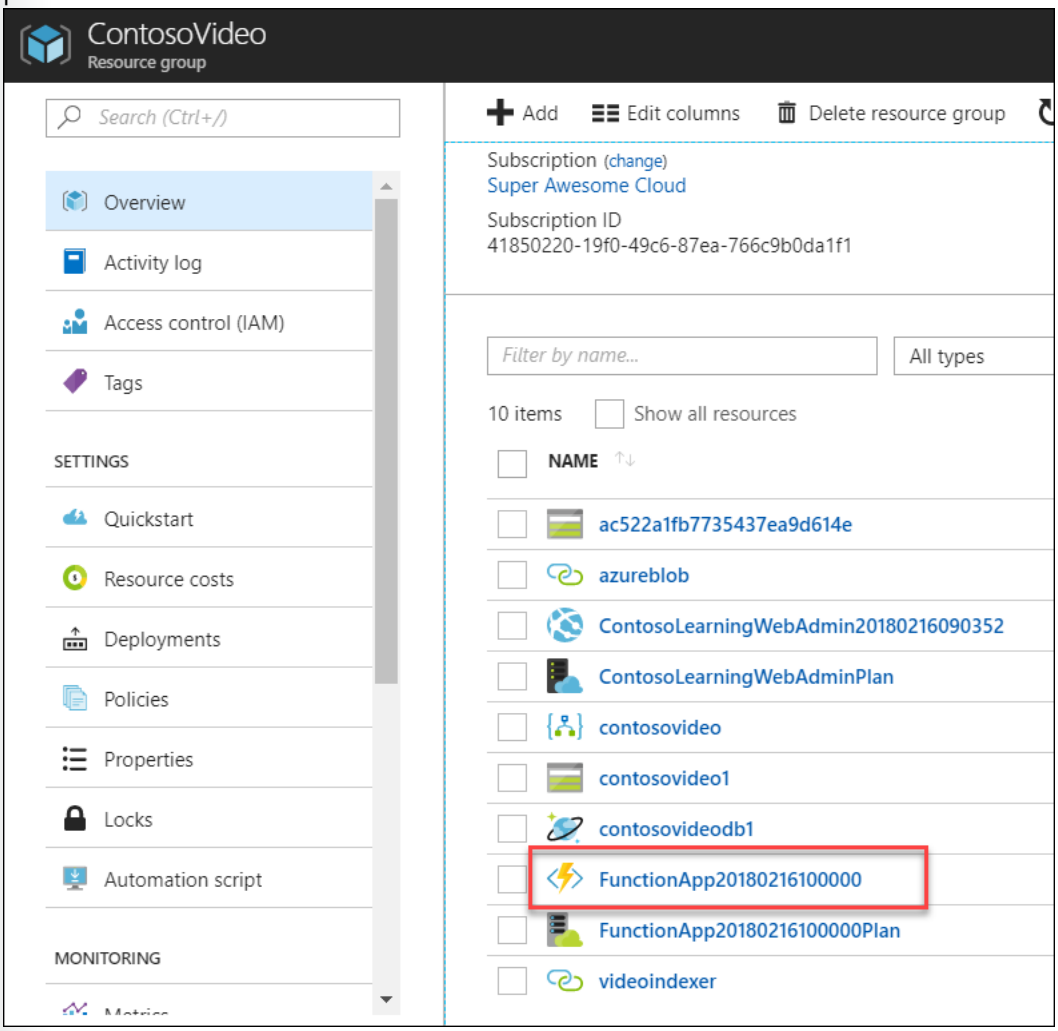
22. Open the **Azure Portal**, and navigate to the **Cosmos DB Account** that was previously created

23. On the **Cosmos DB Account** blade, click on **Keys**

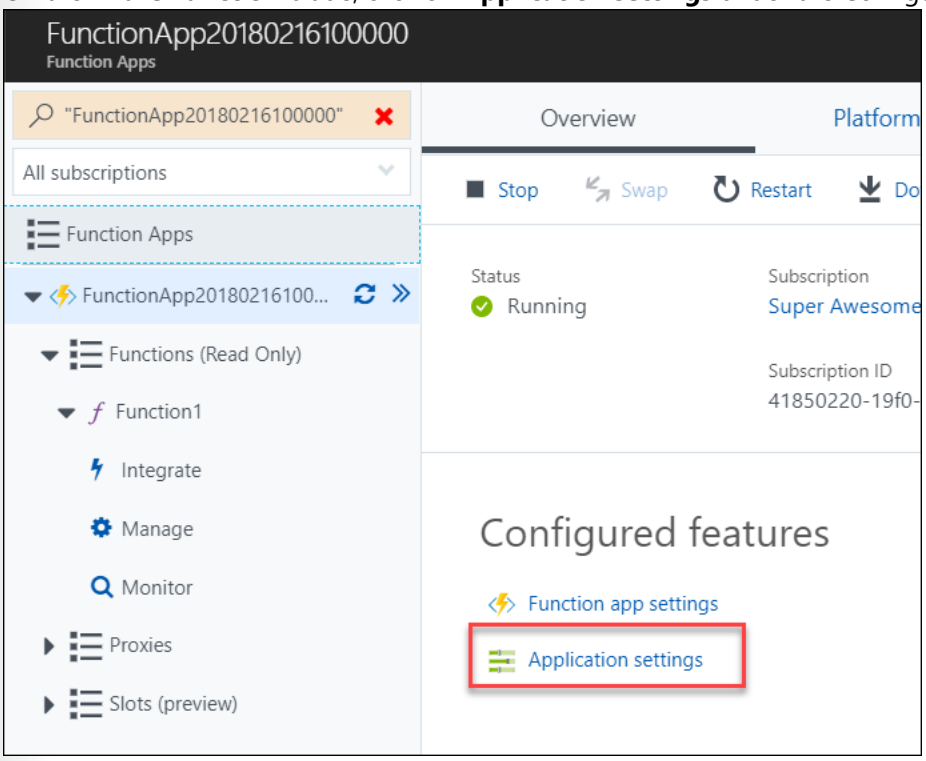


24. Copy the **PRIMARY CONNECTION STRING** for the Cosmos DB Account

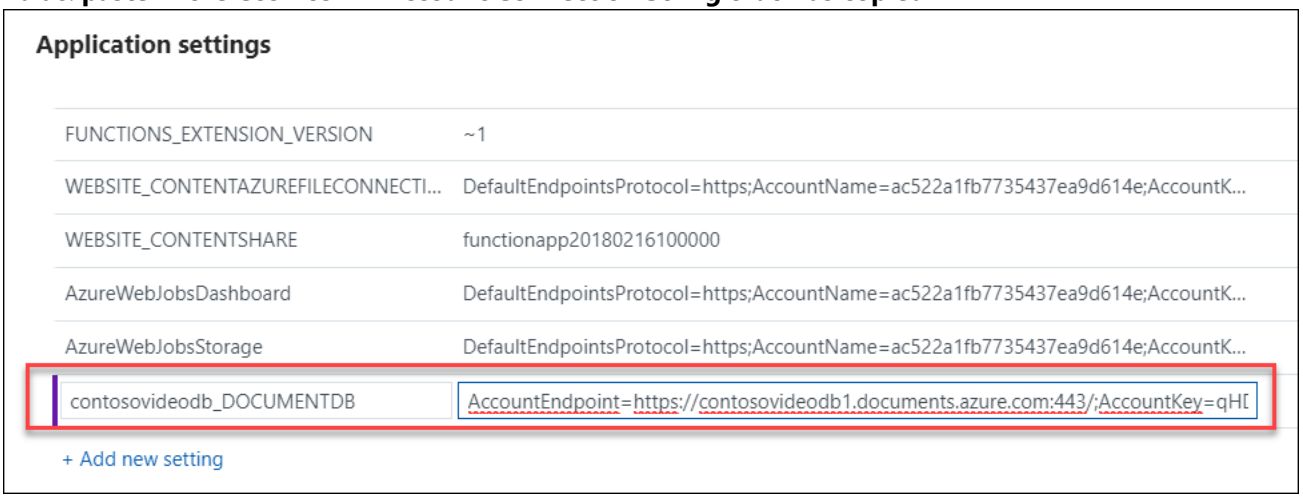

25. Navigate to the **ContosoVideo** Resource Group, then navigate to the **Azure Function** that was created and published from Visual Studio



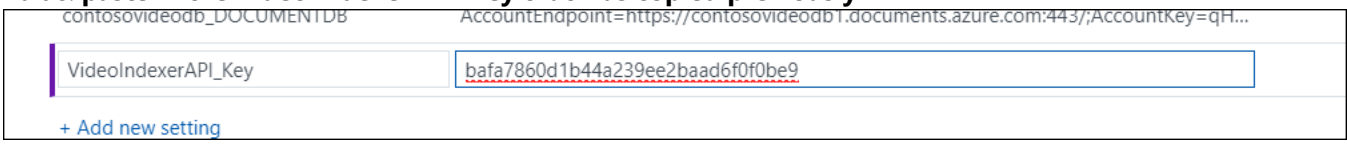
26. On the **Azure Function** blade, click on **Application settings** under the Configured features section



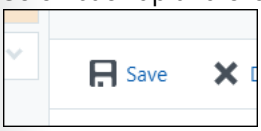

27. Scroll down, and add a new **Application setting** with the following values:
    a. Name: **contosovideodb_DOCUMENTDB**
    b. Value: **paste in the Cosmos DB Account Connection String that was copied**




28. Add another **Application setting** with the following values:
    a. Name: **VideoIndexerAPI_Key**
    b. Value: **paste in the Video Indexer API Key that was copied previously**



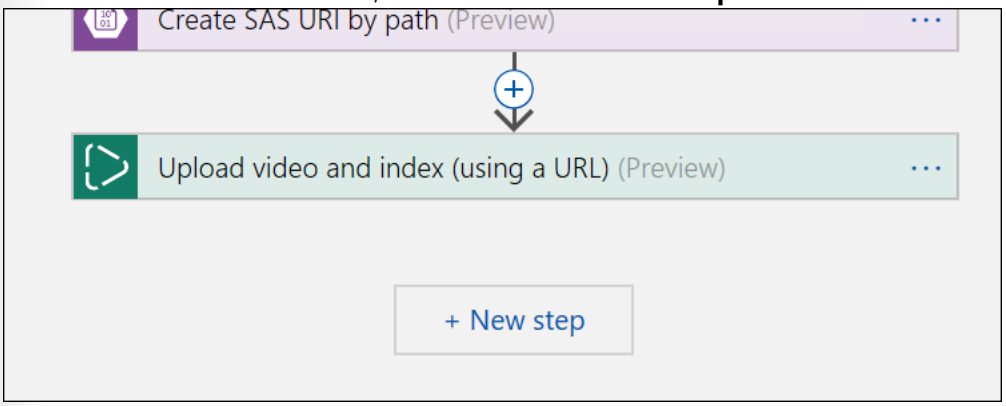
29. Scroll back up and click **Save**



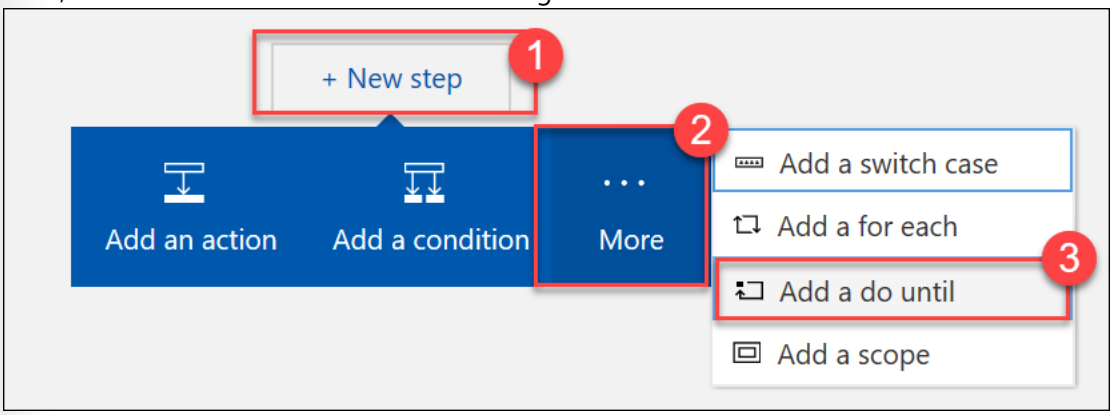

## Step 3: Update Video State when processing is complete

1. Open the **Logic App** within Visual Studio that was previously created

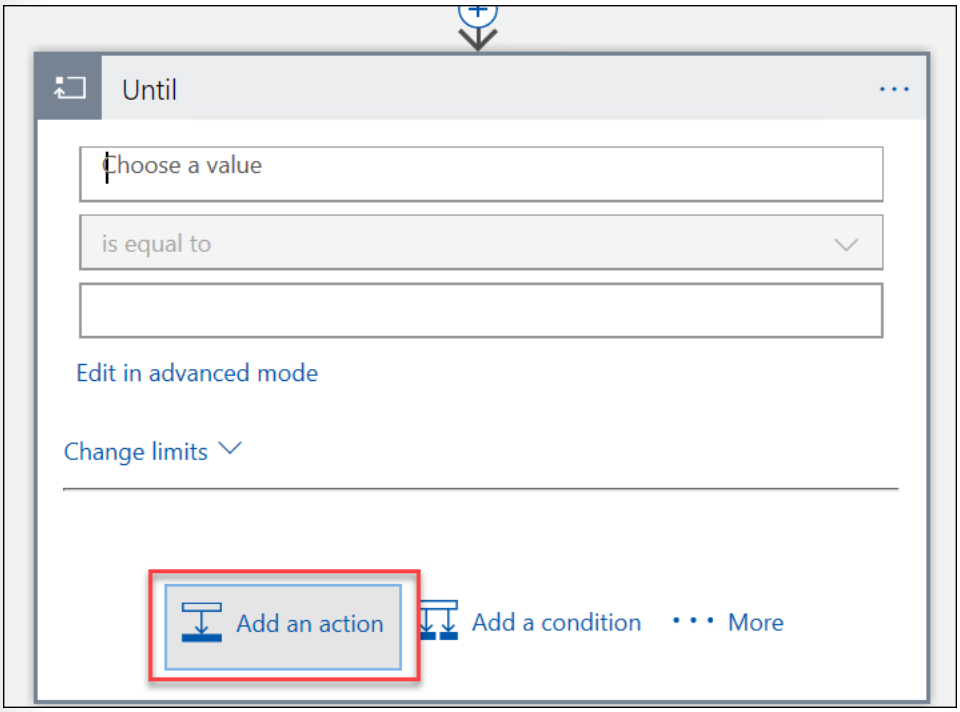
2. Locate the end of the Workflow, after the **Video Indexer – Upload video and index (using a URL)** action



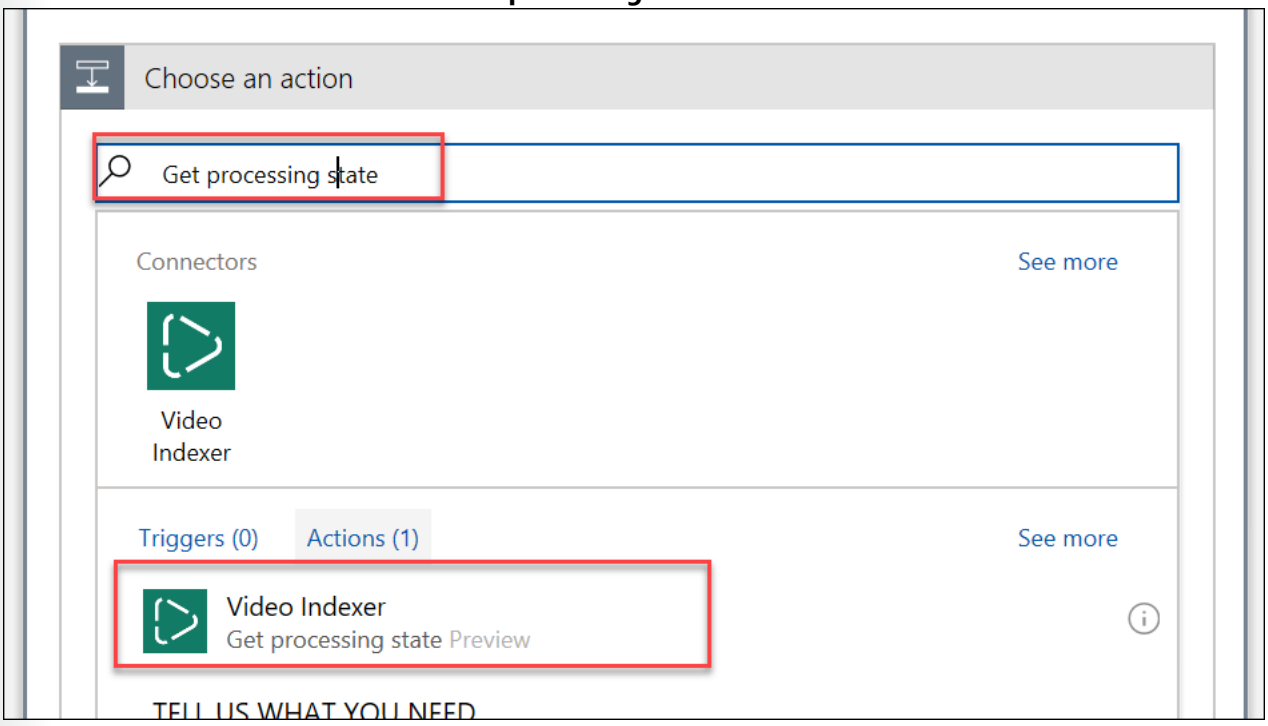

3. Click **+New step**, then **More**, then **Add a do until.** This will be used to periodically check the video processing state, and wait until it's finished before moving on with the workflow.



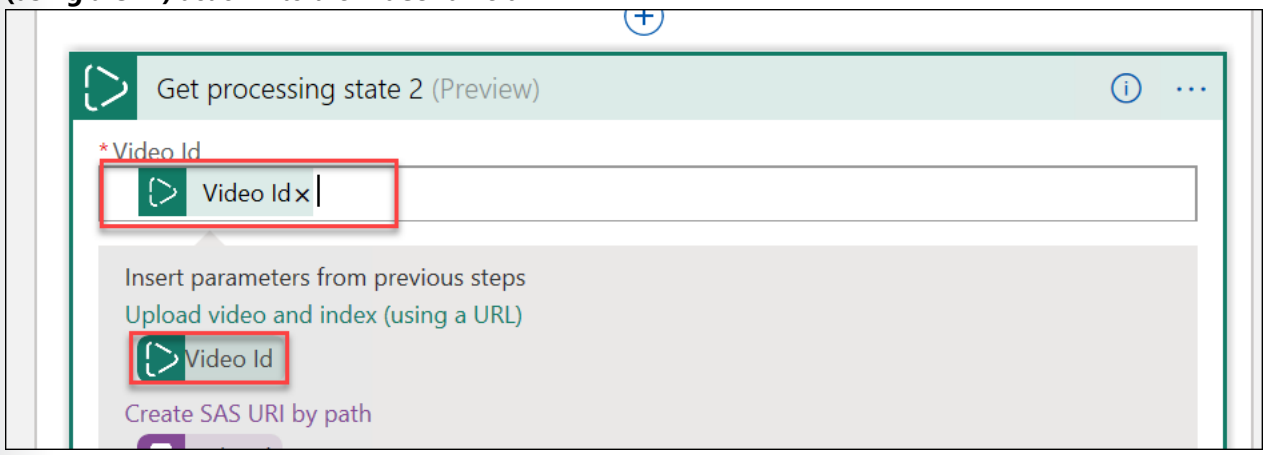
4. Within the **Until** action, click the **Add an action** link to add an action within the "do until" block



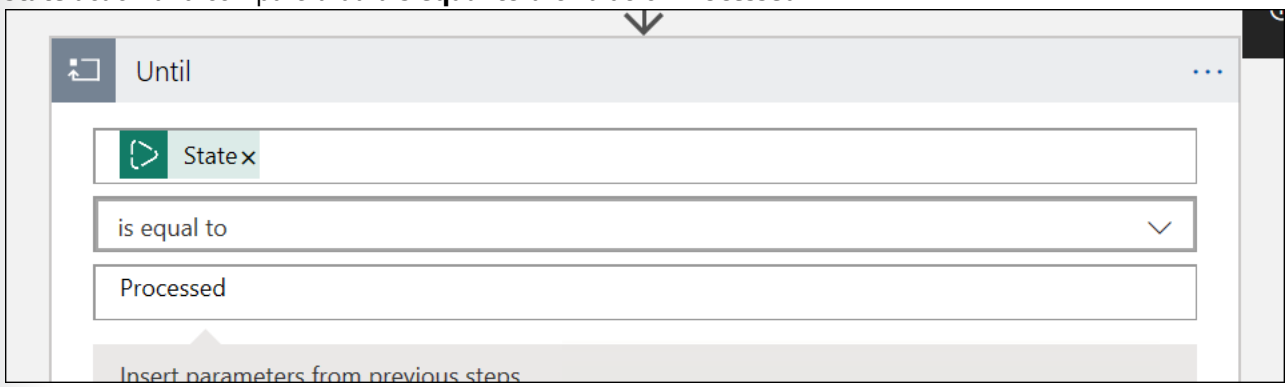

5. Search for and add a **Video Indexer – Get processing state** action



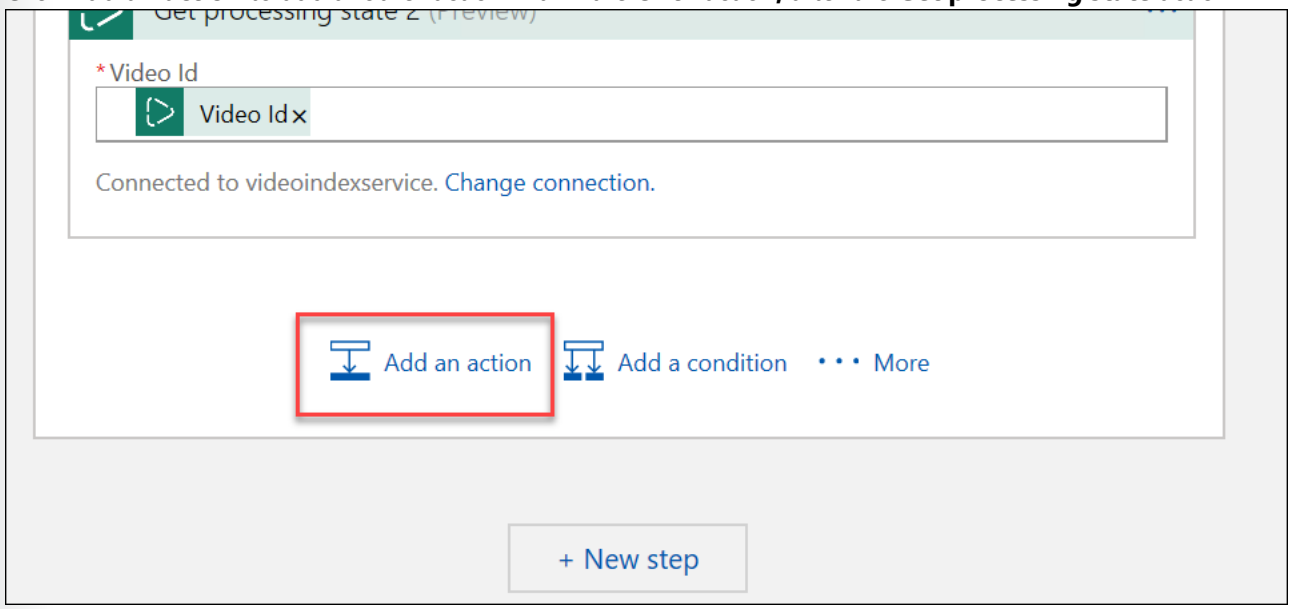
6. On the **Get processing state** action, enter the **Video Id** parameter value from the **Upload video and index (using a URL)** action into the **Video Id** field




7. At the top of the **Until** action, set the check condition to look at the **State** parameter from the **Get processing state** action and compare that it **is equal to** the value of **Processed**

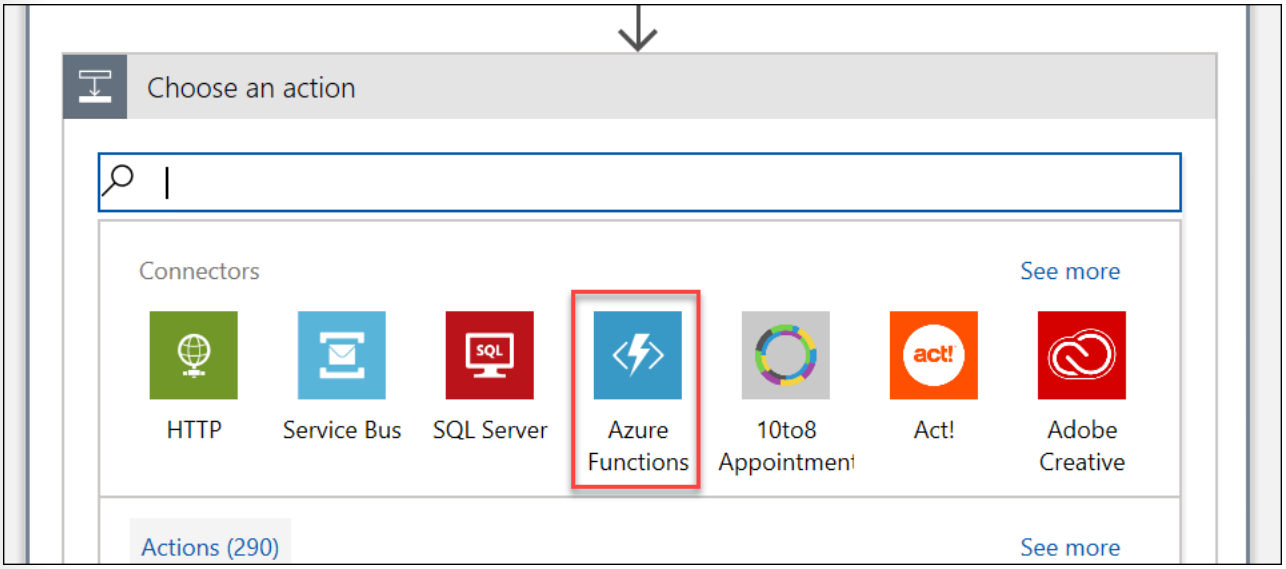



8. Click **Add an action** to add another action within the **Until** action**,** after the **Get processing state** action

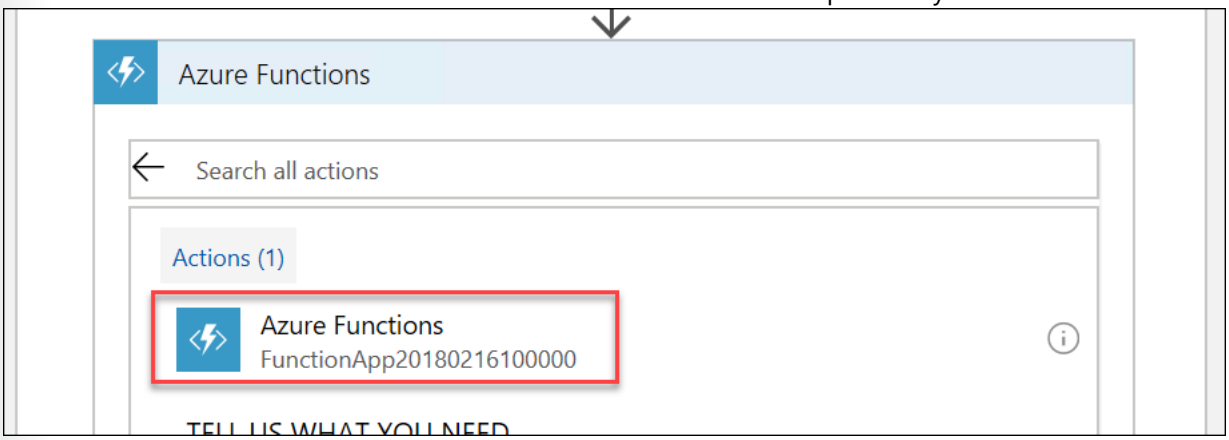


NOTE: The Processing State is being polled here so it can easily be added to the database so the percentage complete can be displayed to the end-user within the web app more easily. Without this feature, the best practice would be to configure a callback with the initial Video Indexer call, so it can asynchronously notify when processing is completed.

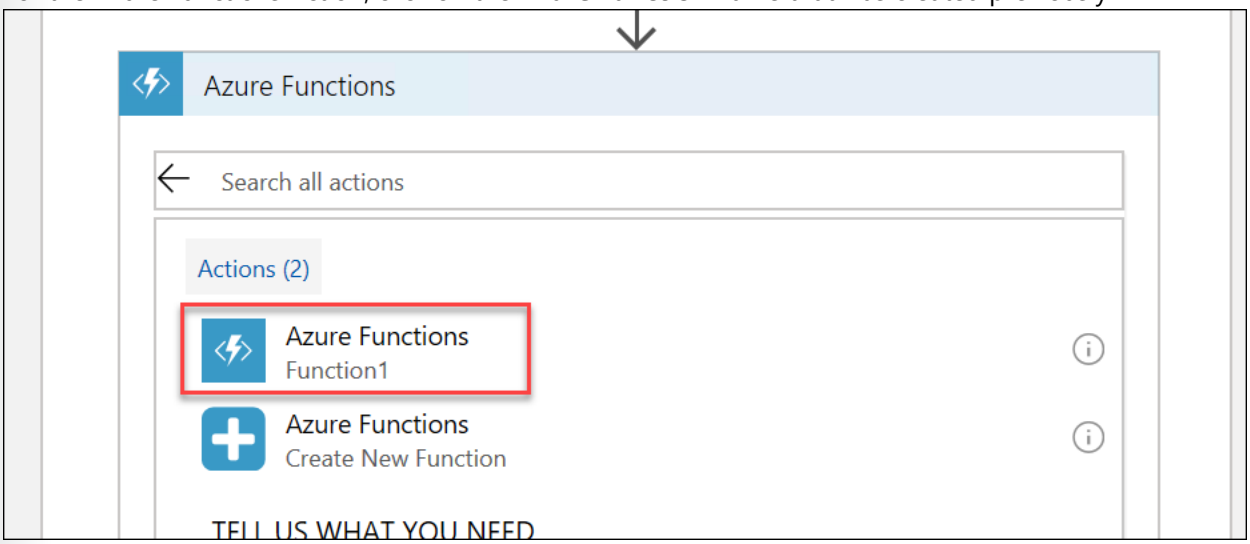
9. Click on the **Azure Functions Connector**




10. Click on the **Azure Functions** Action for the Azure Functions that was previously created

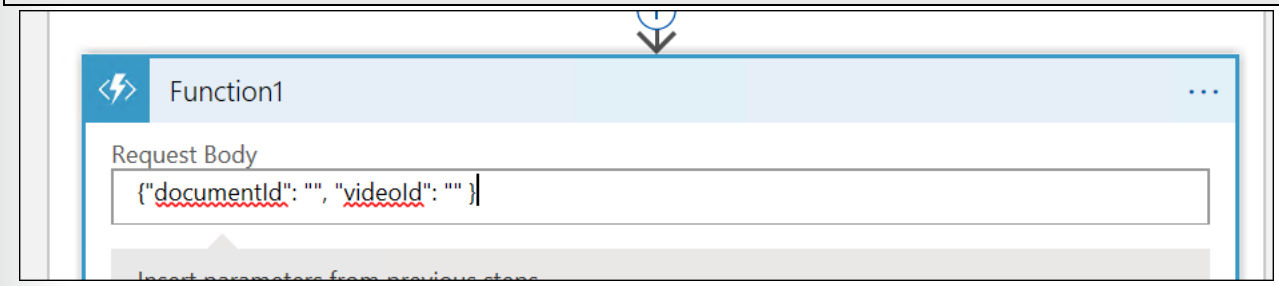


11. For the Azure Functions Action, click on the **Azure Function** name that was created previously



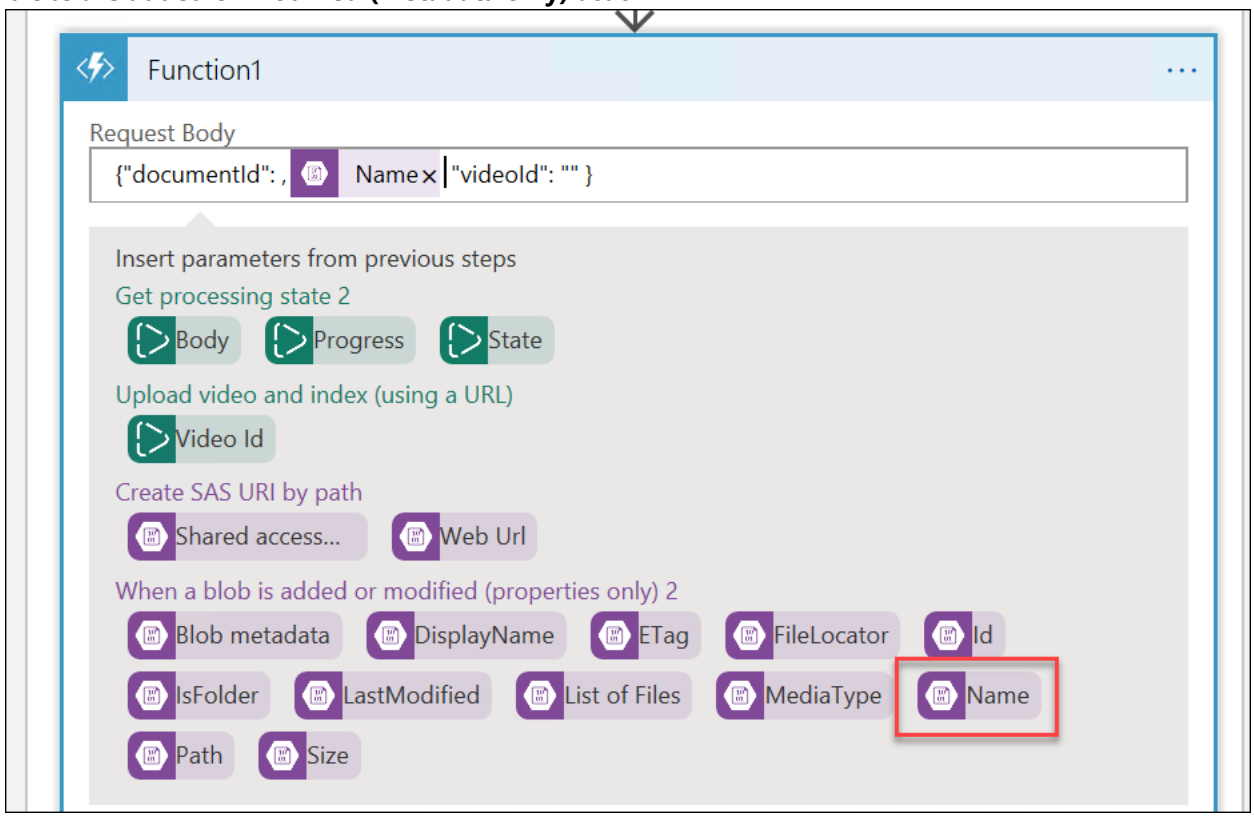

12. Update the **Request Body** field for the Azure Functions Action to contain a JSON object that includes **documentId** and **videoId** values

```
{"documentId": "", "videoId": "" }
```

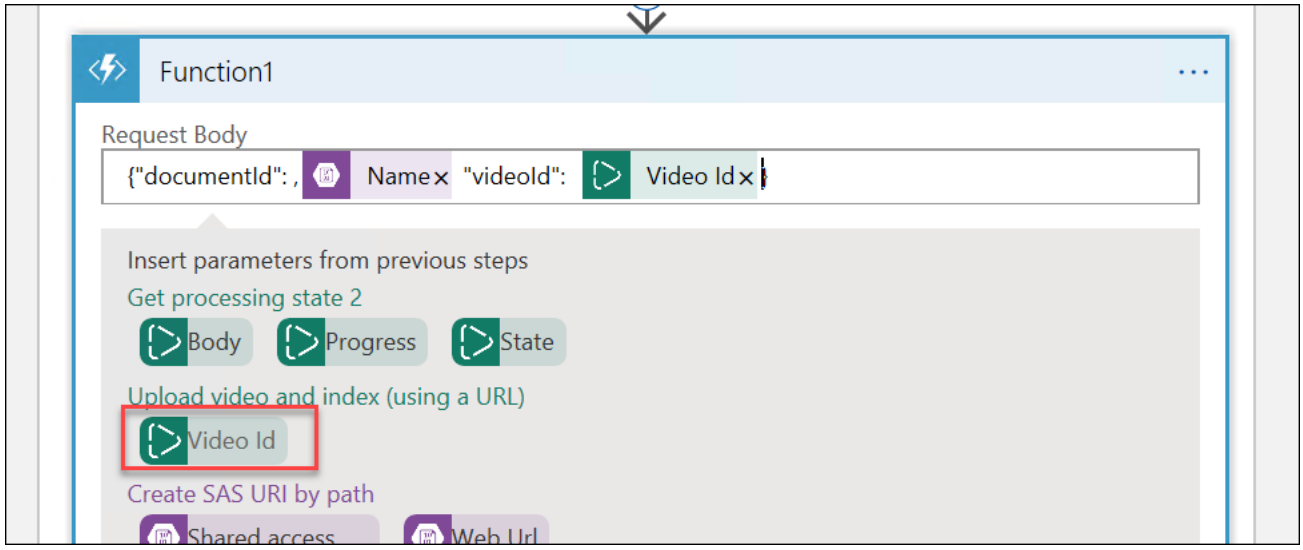



13. Modify the JSON properties to have the following values. Also, be sure to remove the empty double quotes (“”) from the JSON when adding the new property values as shown below.

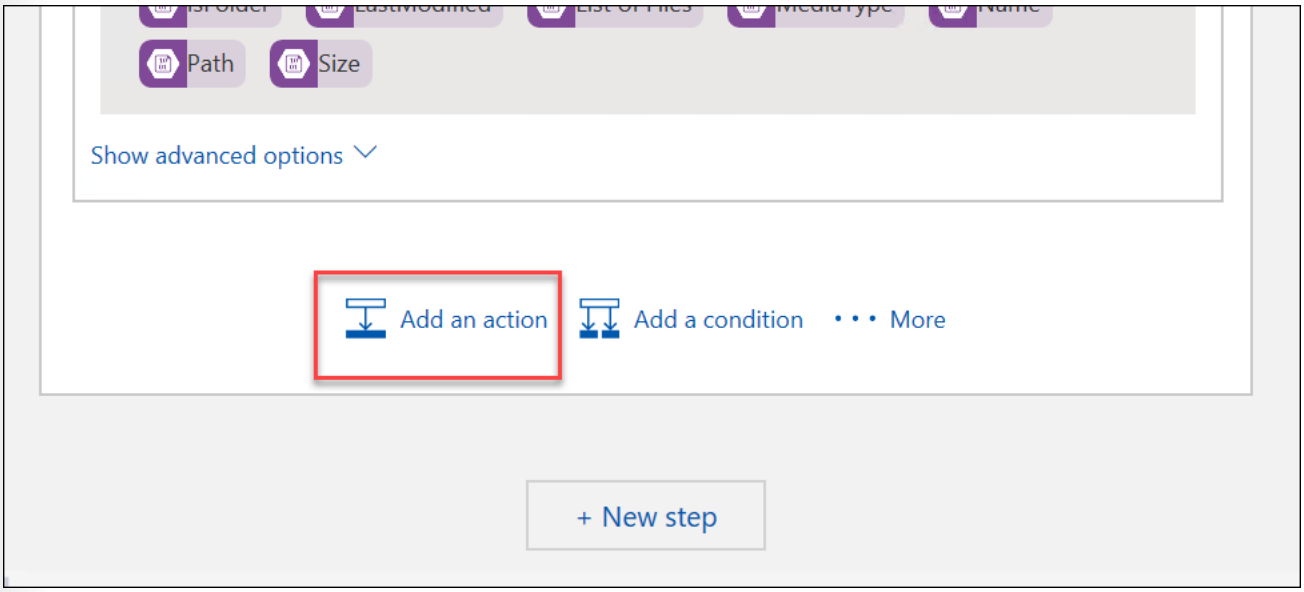
a. Set the **documentId** property to the **Name** parameter from the **Blob Storage – When one or more blobs are added or modified (metadata only)** action



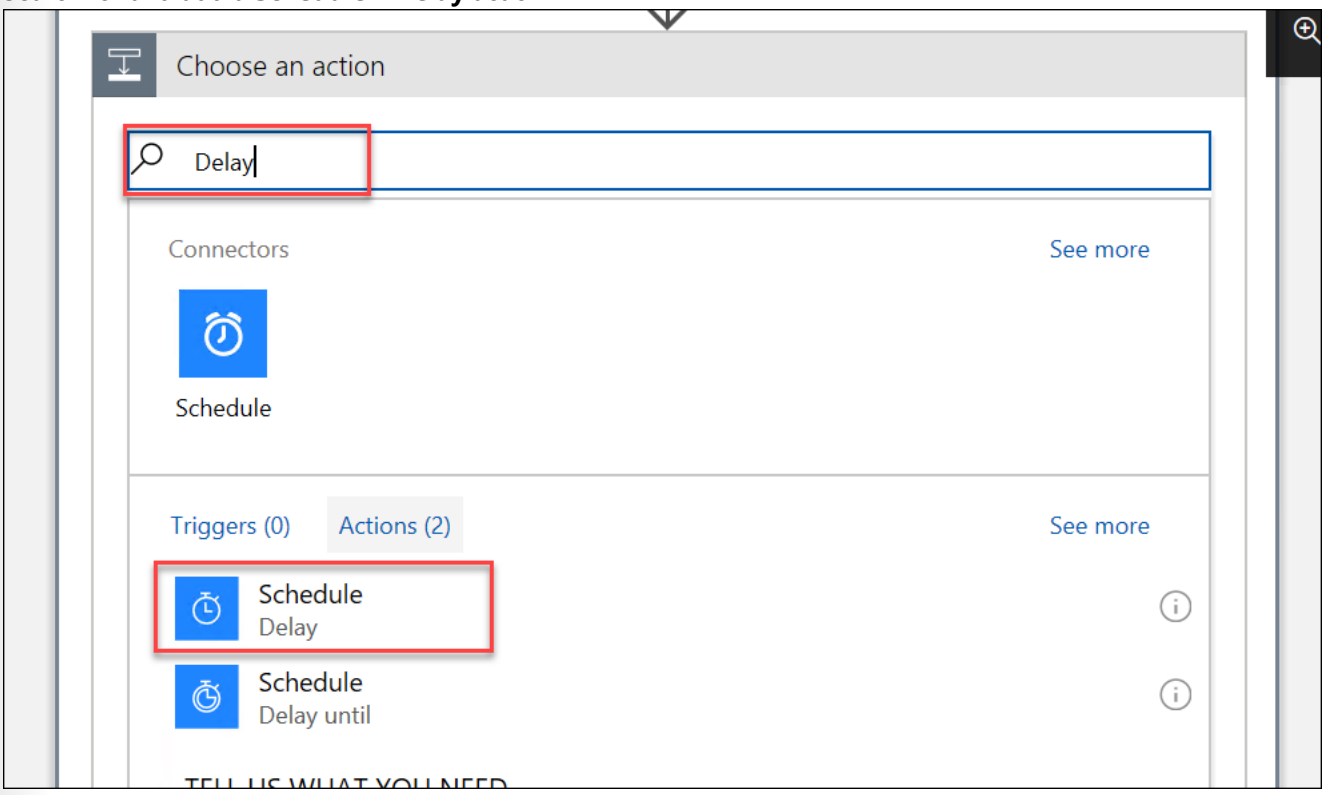

b. Set the **videoId** property to the **Video Id** parameter from the **Video Indexer – Upload and index (using a URL)** action



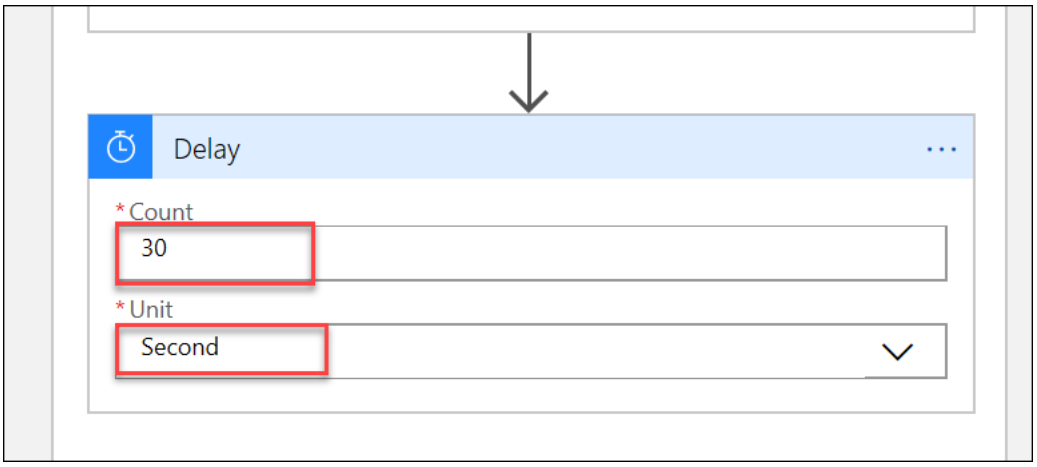
14. Click **Add an action** to add another action within the **Until** action, after the Azure Functions Action



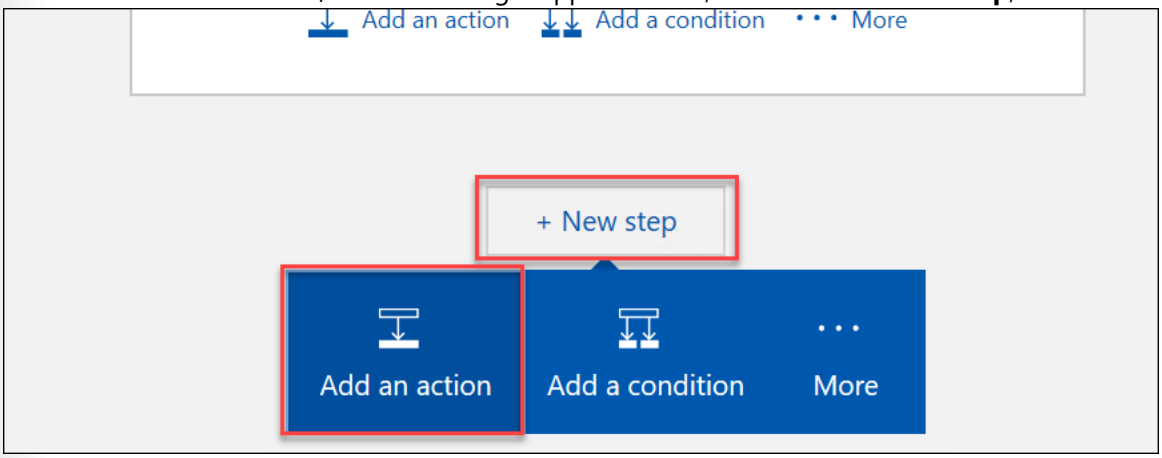

15. Search for and add a **Schedule – Delay** action



16. On the **Delay** action, enter the following values:
    a. Count: **30**
    b. Unit: **Second**




17. Scroll down to the bottom / end of the Logic App Workflow, and click the **+New step**, then click **Add an action**




18. Add another **Azure Functions Action** here that is configured identical to the one previously created within the **Until** loop action. The reason for this is the first one within the **Until** action will periodically update the status of the Video Processing within the Cosmos DB Document with each iteration of the loop. This new Azure Function at the end of the Logic App Workflow will update the Video Processing State one final time before the Workflow finished execution.

19. **Save** the Logic App

20. **Deploy** the updated Logic App to Azure

# Exercise 5: Add Video Player to Front-End Application

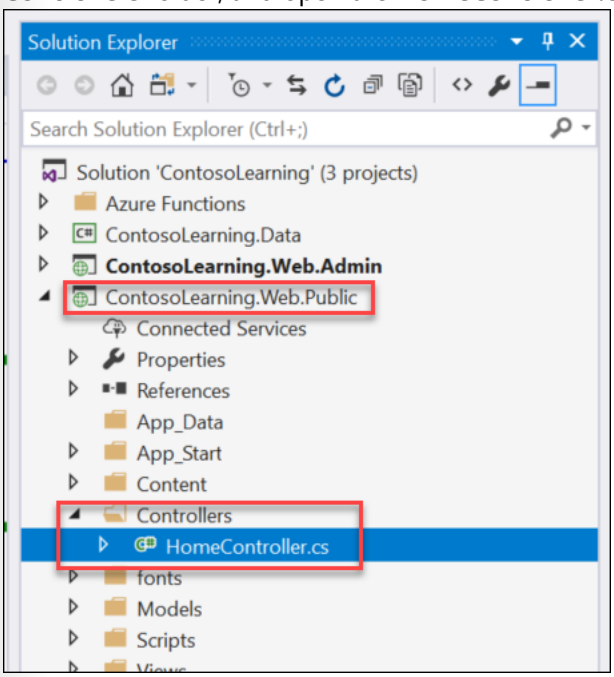
Duration: 30 minutes

In this exercise, you will extend the Front-End Application foundation to include a video player and Cognitive Services Insights for the Videos.

## Step 1: Integrate Cosmos DB into Front-End Application

1. Open the **ContosoLearning.sln** solution within Visual Studio 2017

2. Within Solution Explorer, locate and expand the **ContosoLearning.Web.Public** project, then expand the **Controllers** folder, and open the **HomeController.cs** file




3. Within the **HomeController** class, locate the **Index()** Action method, and replace the methods contents with the following code that uses the **VideoRepository** to load all the Videos from the Cosmos DB Collection and returns the data in the Model so the view can display it in the UI.

```
var model = new HomeIndexModel();

var videoRepo = VideoRepositoryFactory.Create();

model.Videos = (from v in await videoRepo.GetAll()
                    orderby v.Title, v.Created
                    select new VideoListModel
                    {
                        Video = v
                    }).ToArray();

return View(model);
```

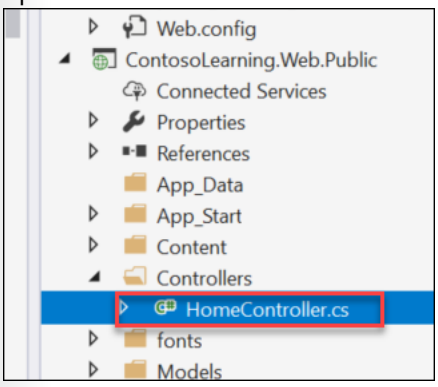
4. Locate the **Video(string id)** method. This Action method is used to display the video player for individual videos. Replace the contents of this method with the following code that loads the Video info from the Cosmos DB Collection.

```
var model = new HomeVideoModel();

var courseRepo = VideoRepositoryFactory.Create();
model.Video = await courseRepo.Get(id);

if (model.Video == null)
{
    throw new Exception("Video not found!");
}

// Get Access Token
var client = new System.Net.Http.HttpClient();
client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key",
System.Configuration.ConfigurationManager.AppSettings["VideoIndexerAPI_Key"]);

var uriResponse = await client.GetAsync($"https://videobreakdown.azure-
api.net/Breakdowns/Api/Partner/Breakdowns/{model.Video.VideoId}/GetAccessToken");
var jsonUriResponse = await uriResponse.Content.ReadAsStringAsync();

model.AccessToken = jsonUriResponse.Replace("\"", string.Empty);



return View(model);
```

5. **Save** the file.

## Step 2: Display Video Thumbnail Image

1. Within Solution Explorer, expand the **Controllers** folder within the **ContosoLearning.Web.Public** project, then open the **HomeController.cs** file




2. Locate the **Index()** action method within the **HomeController** class

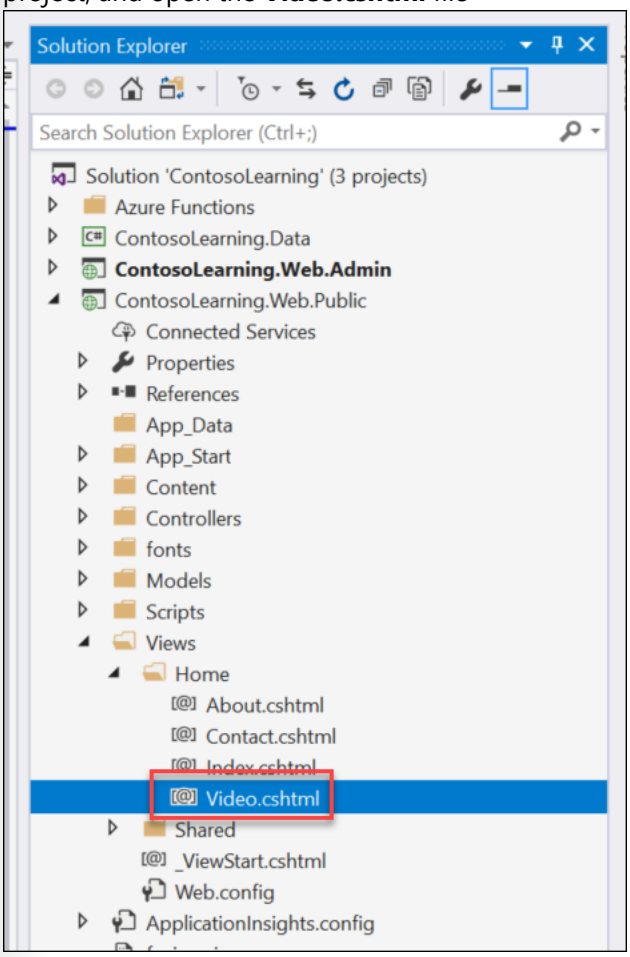
 

3. Paste in the following code immediately before the **return** statement in the **Index()** method. This code will loop through all the Videos and load the **Video Thumbnail URL** for each by calling the **Video Indexer API**.

```
var client = new System.Net.Http.HttpClient();
var queryString = HttpUtility.ParseQueryString(string.Empty);

// Request headers
client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key",
System.Configuration.ConfigurationManager.AppSettings["VideoIndexerAPI_Key"]);

// Request parameters
queryString["language"] = "{string}";

foreach (var v in model.Videos) {
    var uri = $"https://videobreakdown.azure-
api.net/Breakdowns/Api/Partner/Breakdowns/{v.Video.VideoId}";
    var response = await client.GetAsync(uri);
    var json = await response.Content.ReadAsStringAsync();

    dynamic breakdown =  Newtonsoft.Json.Linq.JObject.Parse(json);

    v.ThumbnailUrl = breakdown?.summarizedInsights?.thumbnailUrl;
}
```

## Step 3: Add Video Player

1. Within Solution Explorer, locate and expand the **Views/Home** folder within the **ContosoLearning.Web.Public** project, and open the **Video.cshtml** file




2. Locate the "*[Video Here]*" placeholder text within the **Video.cshtml** view

```
18          <div class="col-lg-6">
19
20              [Video Here]
21
22          </div>
```

3. Replace the placeholder text with the following code that will include the **Video Player** within an IFrame. Notice the **VideoId** property from the Video is appended to the URL within the IFrame to tell Video Indexer which video to play.

```
<iframe width="560" height="315"
src="https://www.videobreakdown.com/embed/player/@(Model.Video.VideoId)?accessToken=@(Model.AccessTok
en)" frameborder="0" allowfullscreen></iframe>
```

## Step 4: Add Video Insights

1. Within the **Video.cshtml** file, locate the "*[Insights Here]*" placeholder text

```
23          <div class="col-lg-6">
24
25              [Insights Here]
26
27          </div>
```

2. Replace the placeholder text with the following code that will include the **Video Insights** within an IFrame. Notice the **VideoId** property from the Video is appended to the URL within the IFrame to tell Video Indexer which video to display Insights for.

```
<iframe style="width: 100%; height: 60em;"
src="https://www.videobreakdown.com/embed/insights/@(Model.Video.VideoId)?accessToken=@(Model.AccessT
oken)" frameborder="0" allowfullscreen="true"></iframe>
```

## Step 5: Integrate Video Player and Insights together

1. As coded previously, the Video Player and Insights will display, but are disconnected

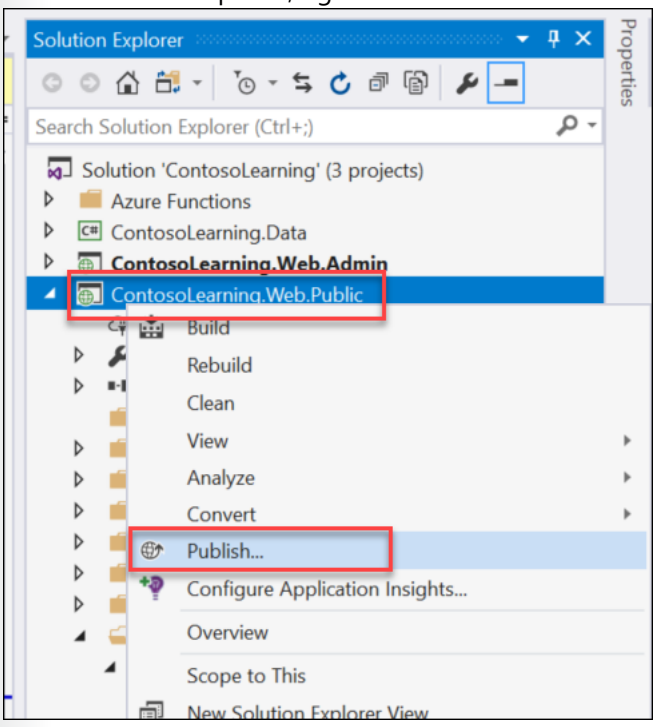
2. To connect the Video Player and Insights so the insights can update as the video plays, add the following JavaScript file include to the bottom of the **Video.cshtml** View below the IFrame code:

```
<script src="https://breakdown.blob.core.windows.net/public/vb.widgets.mediator.js"></script>
```

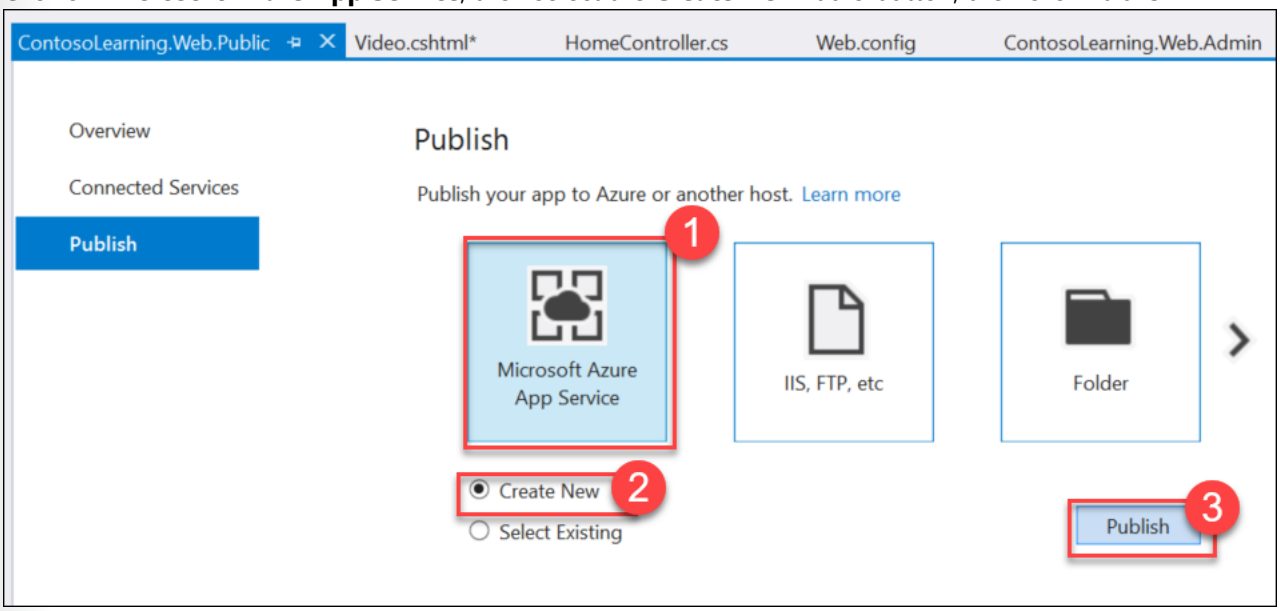
3. Save the file

## Step 6: Deploy Public website to an Azure Web App

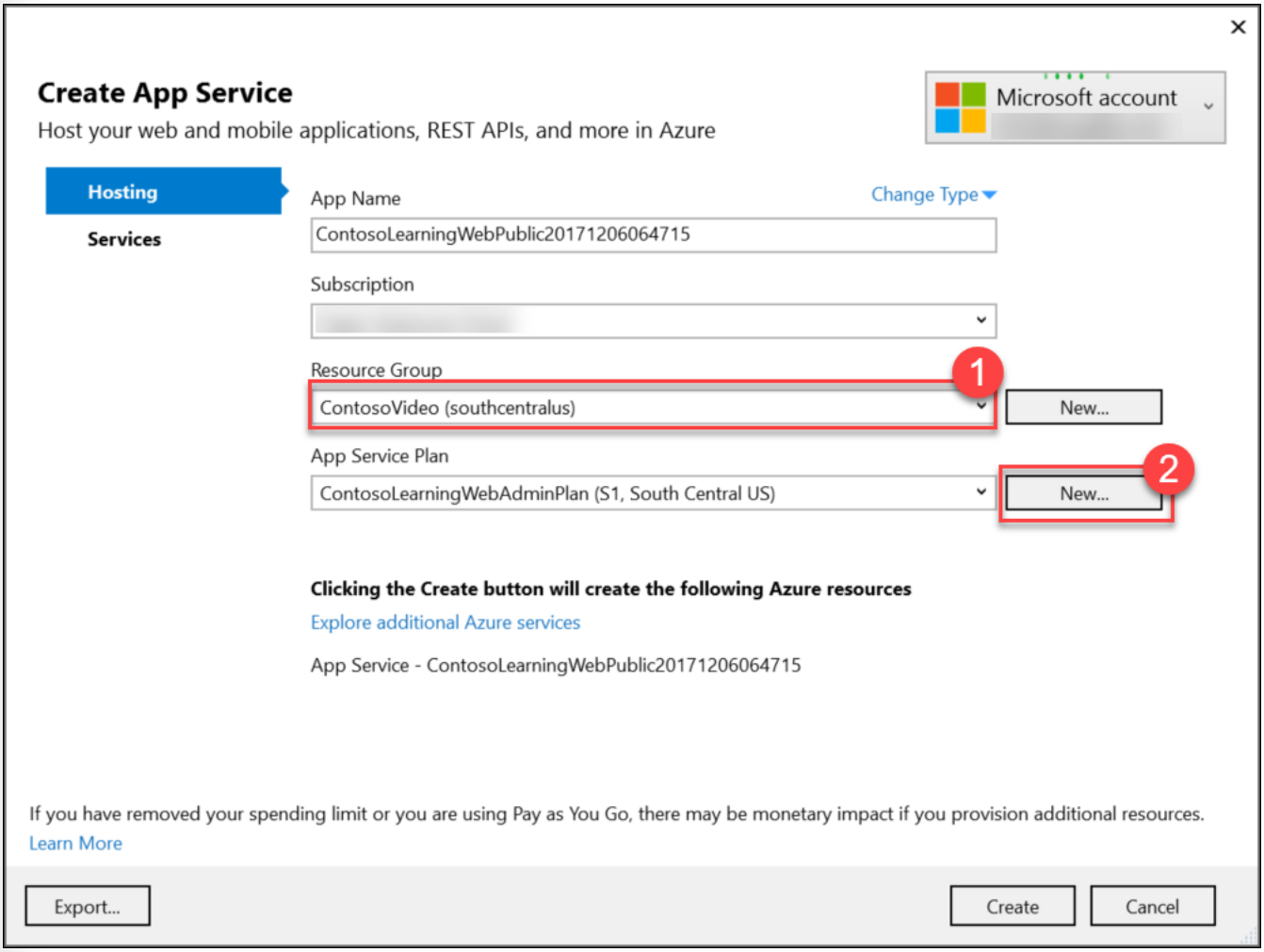
1. Within Solution Explorer, right-click the **ContosoLearning.Web.Public** project, then click **Publish...**



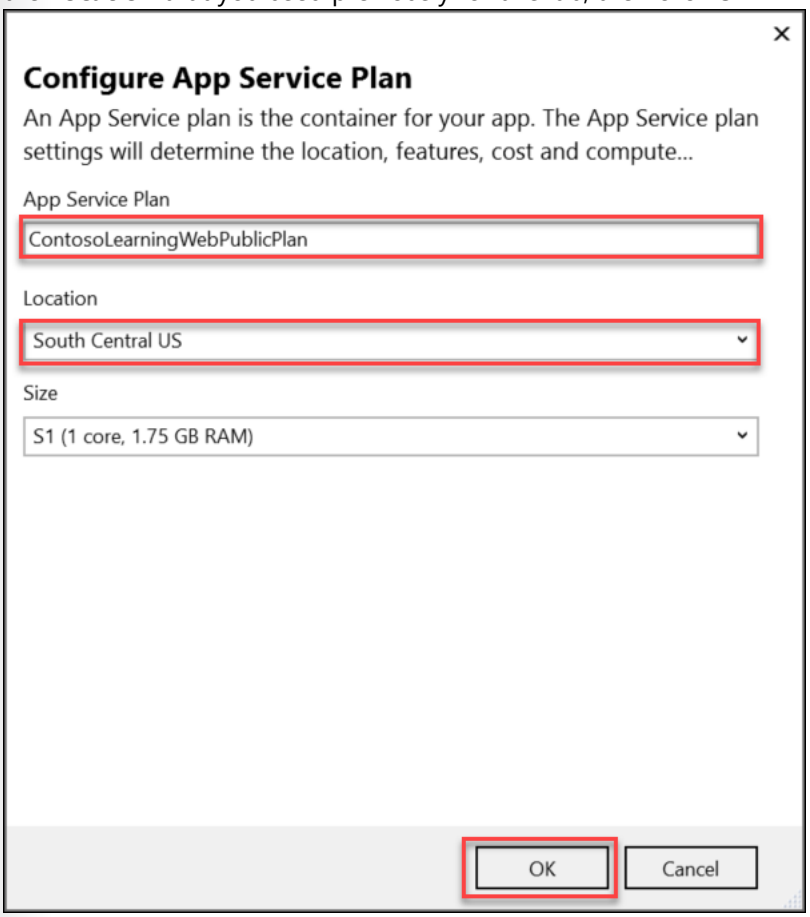
2. Click on **Microsoft Azure App Service**, then select the **Create New** radio button, then click **Publish**



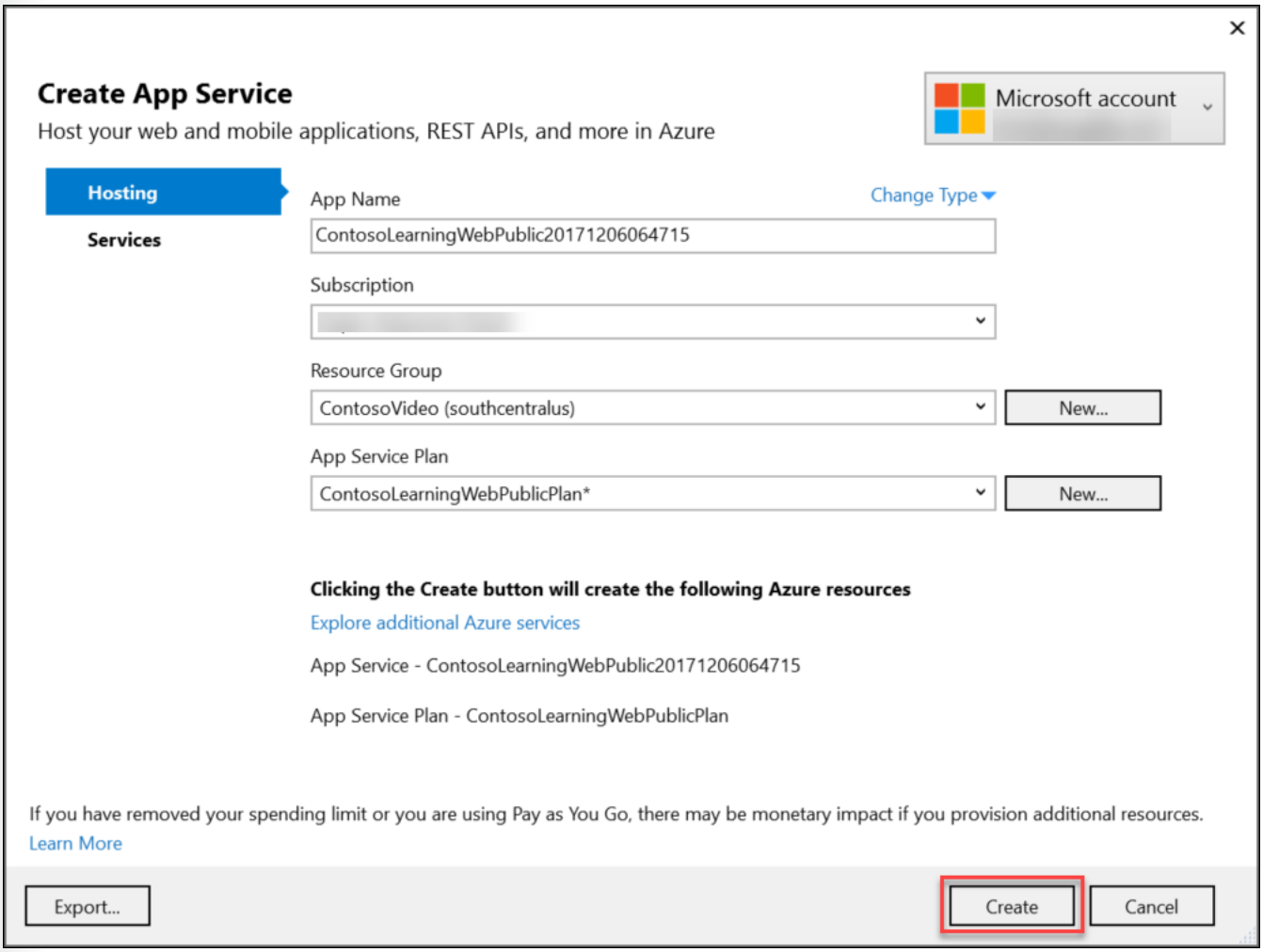

3. On the **Create App Service** dialog, select the **ContosoVideo** Resource Group, then click **New** next to **App Service Plan**



4. On the **Configure App Service Plan** dialog, enter a valid name into the **App Service Plan** name field, then select the **Location** that you used previously for this lab, then click **OK**

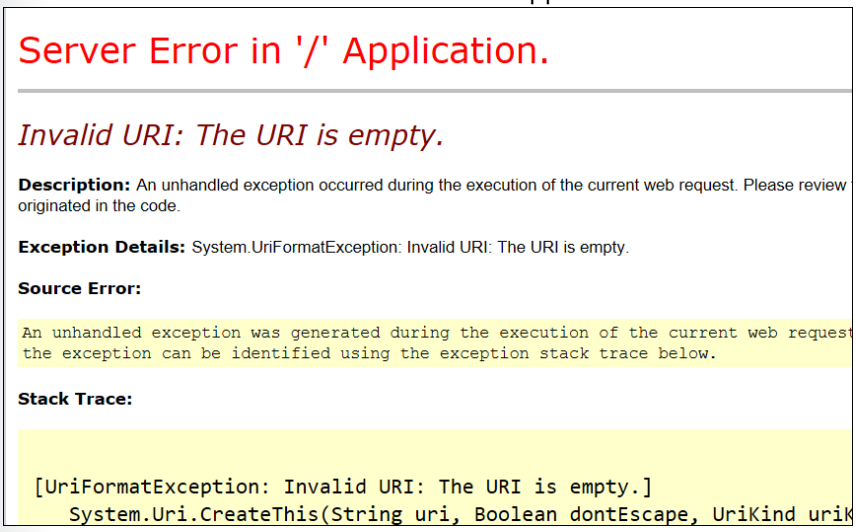


5. Click **Create**



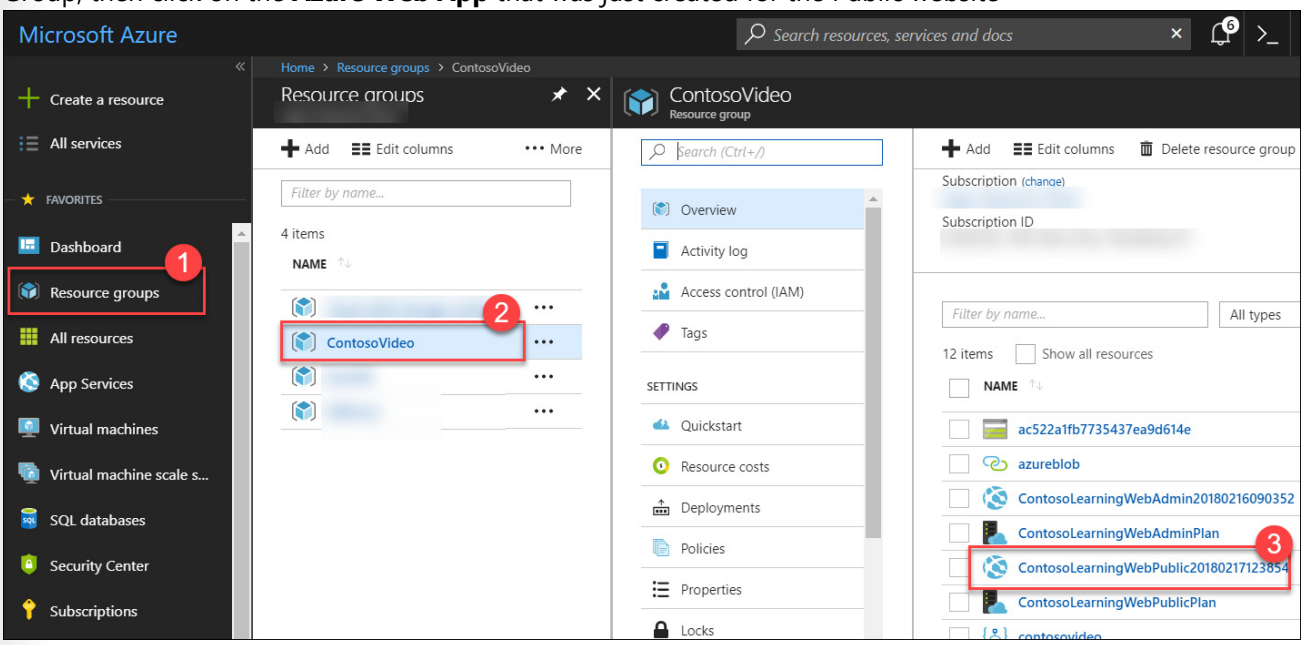

6. Once the application has finished deploying to the Azure Web App, a new browser window will be opened and navigated to the web app running in Azure

7. You should now see an Error screen for the app. This is normal since the AppSettings haven't been configured yet.

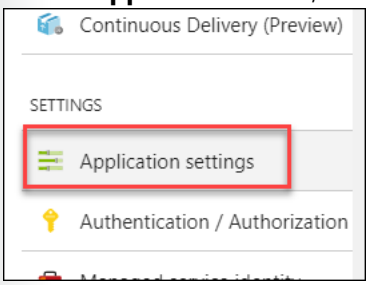


## Step 7: Configure Application Settings

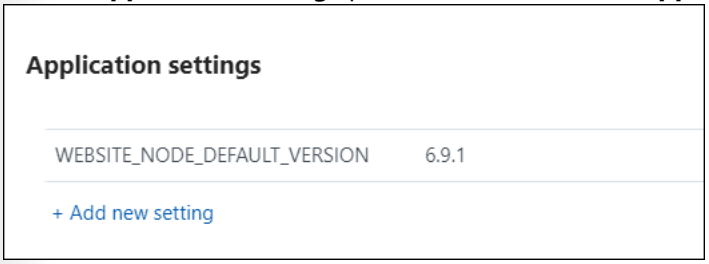
1. Open the **Azure Portal**, click on **Resource groups** in the left-side menu, then click the **ContosoVideo** Resource Group, then click on the **Azure Web App** that was just created for the Public website




2. On the **App Service** blade, click on **Application settings**



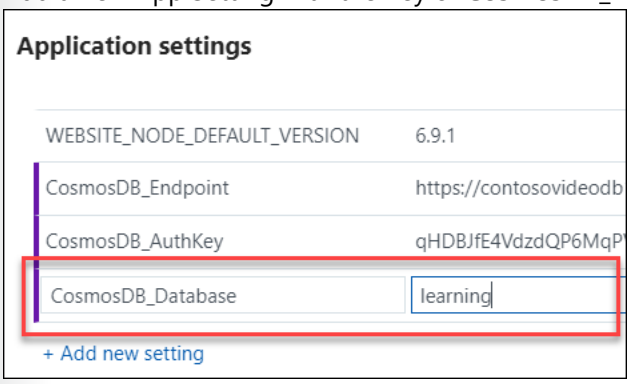

3. On the **Application settings** pane, scroll down to the **App settings** section



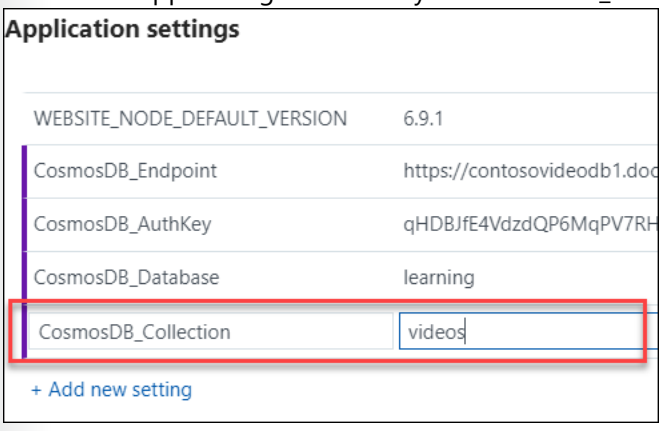
4. Add a new App Setting with the Key of **CosmosDB_Endpoint** with the Value set to the **Cosmos DB Account URI** that was copied


5. Add a new App Setting with the Key of **CosmosDB_AuthKey** with the Value set to the **Cosmos DB Account Primary Key** that was copied

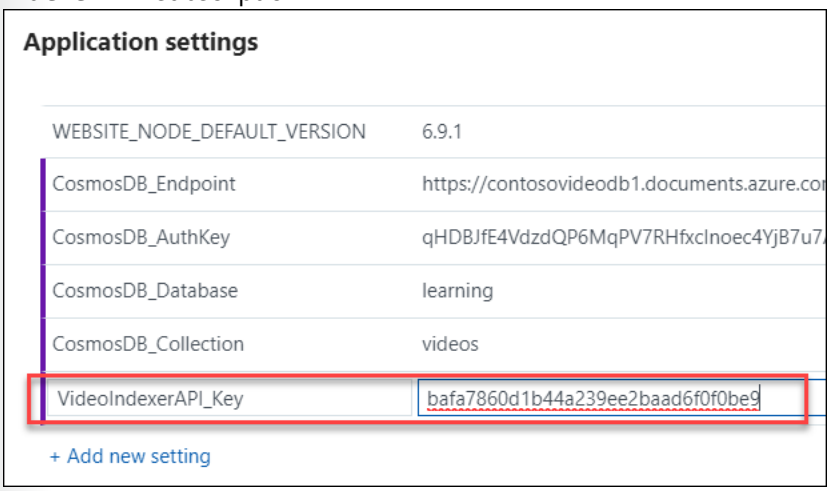

6. Add a new App Setting with the Key of **CosmosDB_Database** with the Value set to **learning**



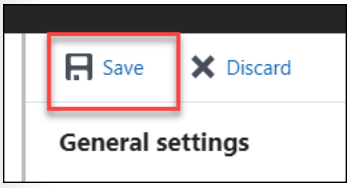

7. Add a new App Setting with the Key of **CosmosDB_Collection** with the Value set to **videos**



8. Add a new App Setting with the Key of **VideoIndexerAPI_Key** with the Value set to the **API Key** for the **Video Indexer API** subscription




9. Click **Save**



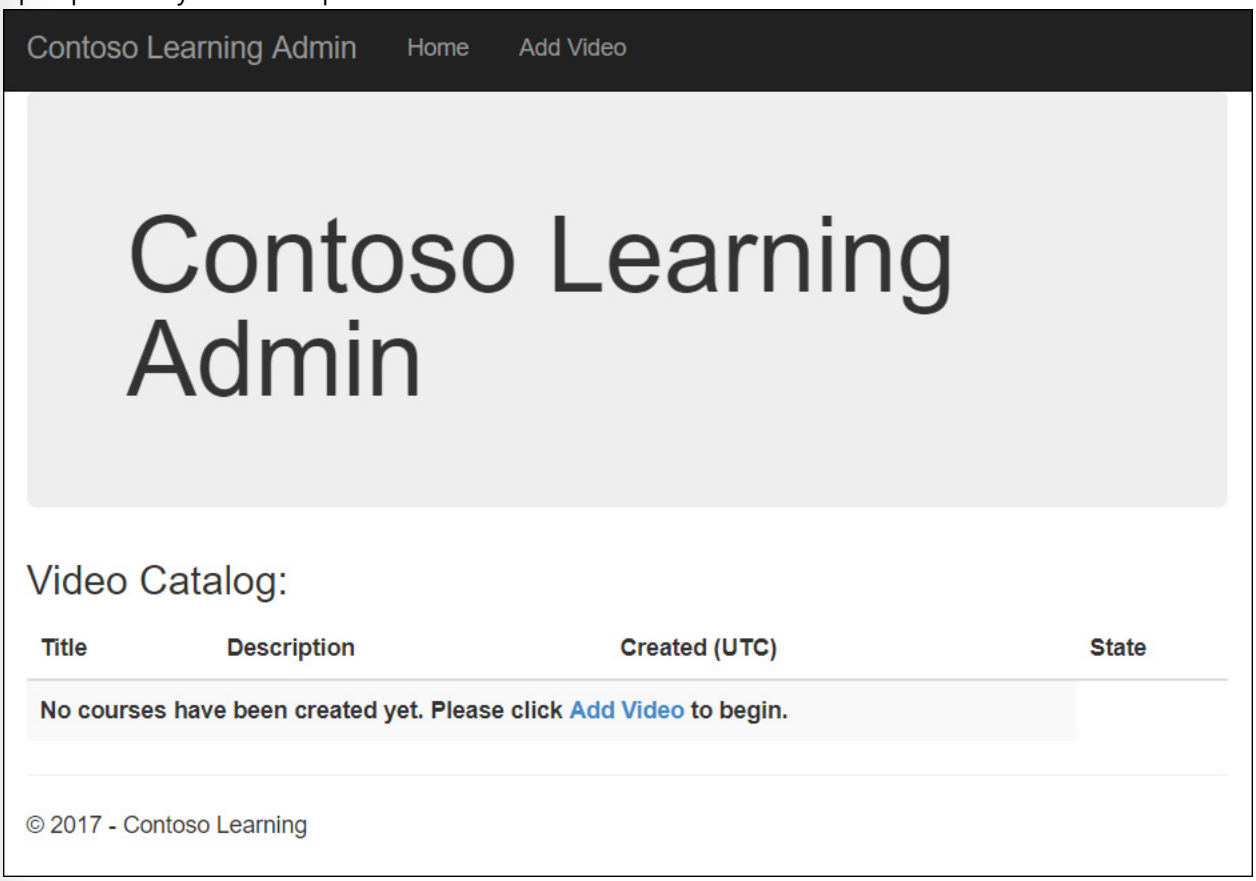
# Exercise 6: Test the application

Duration: 15 minutes

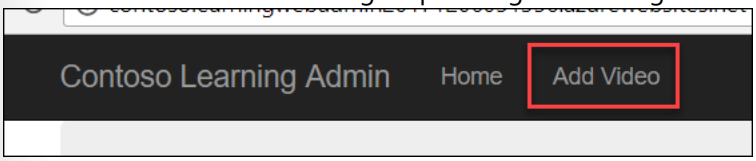
In this exercise, you will test out the Admin and Public web applications.

## Step 1: Upload Video to Admin website

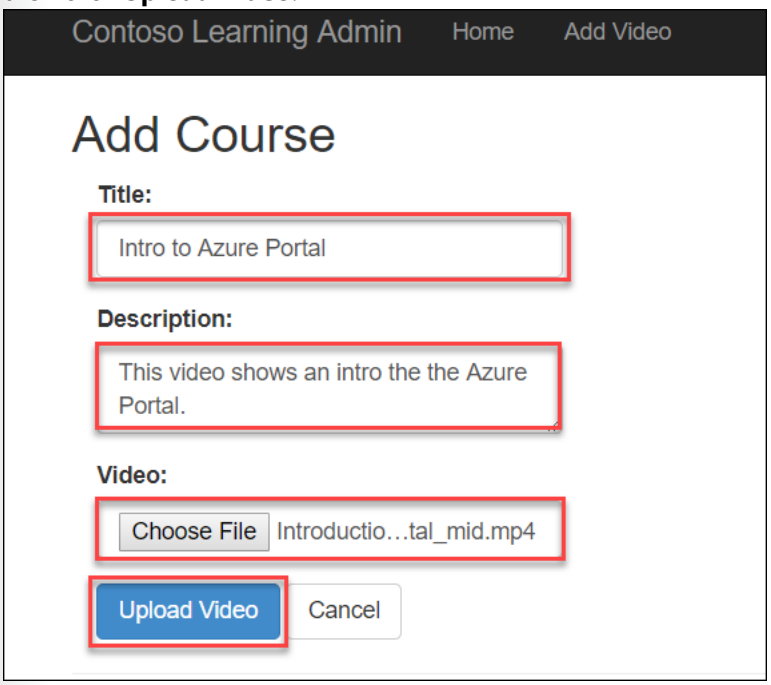
1. Open a browser window to the **Admin Website** running in Azure Web Apps. You can use the window that was open previously if it's still open




2. Click the **Add Video** link to begin uploading and adding a new Video to the catalog in the application

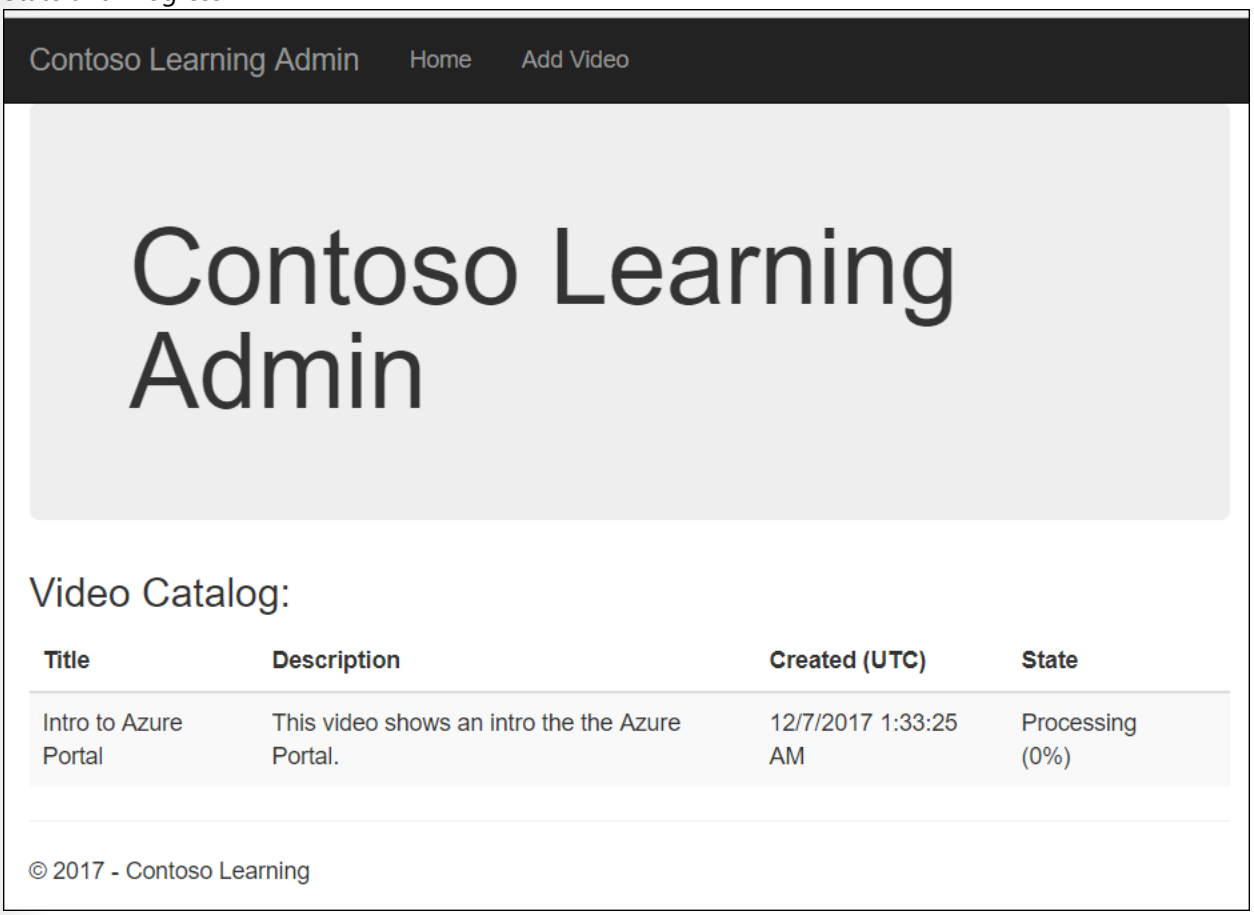


3. Enter some text into the form for the **Title** and **Description** fields, then click to choose a **Video** file to upload. Choose the "*Introduction-to-the-Azure-Portal_mid.mp4*" video file downloaded with the Student Files for this Lab, then click **Upload Video**.




NOTE: Uploading the video file may take a few minutes depending on your Internet connection.

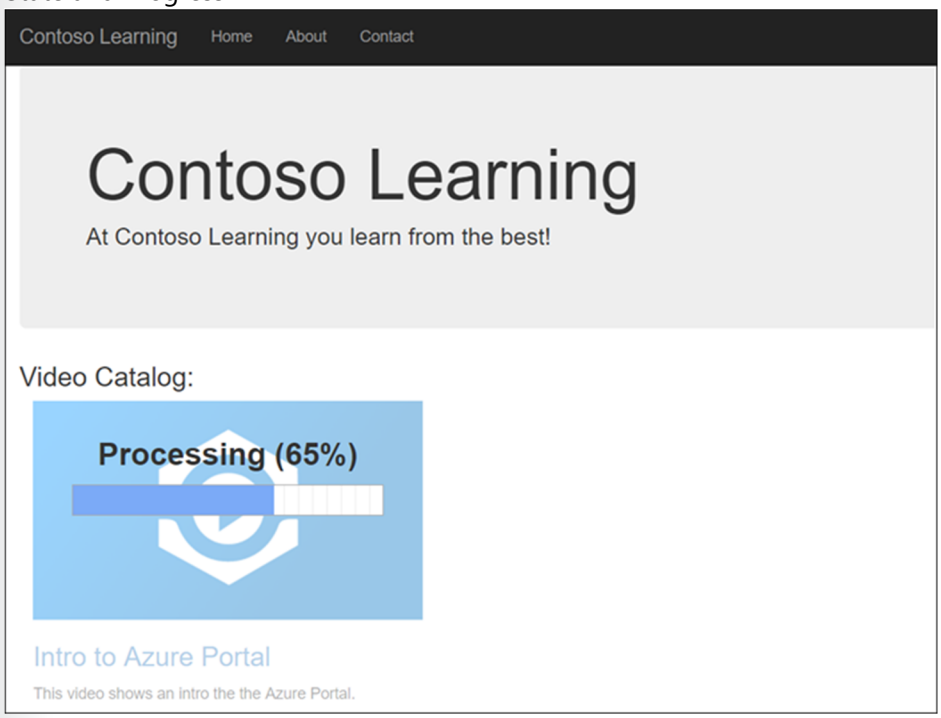
4. Once the video file has been uploaded, the homepage of the Admin app will load displaying the Video Processing State and Progress



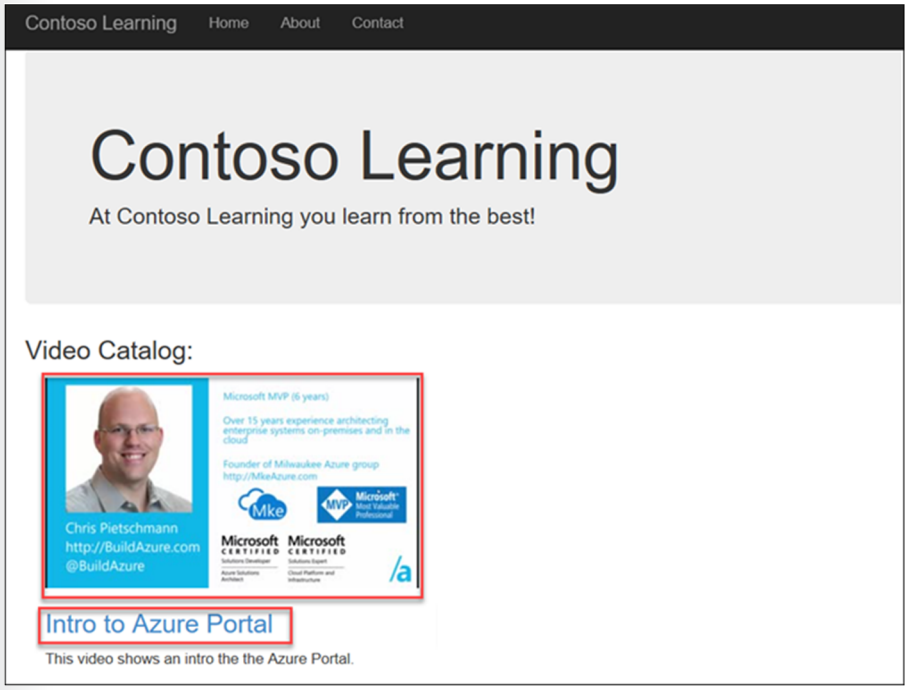

## Step 2: View Video and Insights in Public website

1. Open a browser window to the **Public Website** running in Azure Web Apps. If using the window previously opened, just refresh the page to reload it.

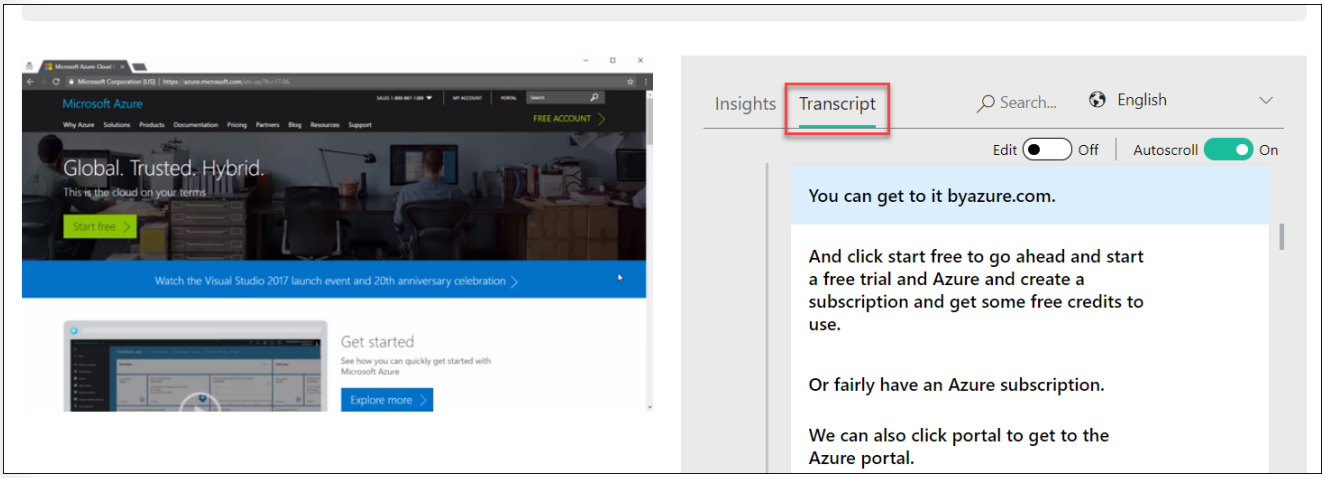
2. While the Videos are Processing within the **Video Indexer** service, the Public Website will display the Processing State and Progress




3. Once the videos have finished processing, click on the Thumbnail image or the Title of the Video to view that Video

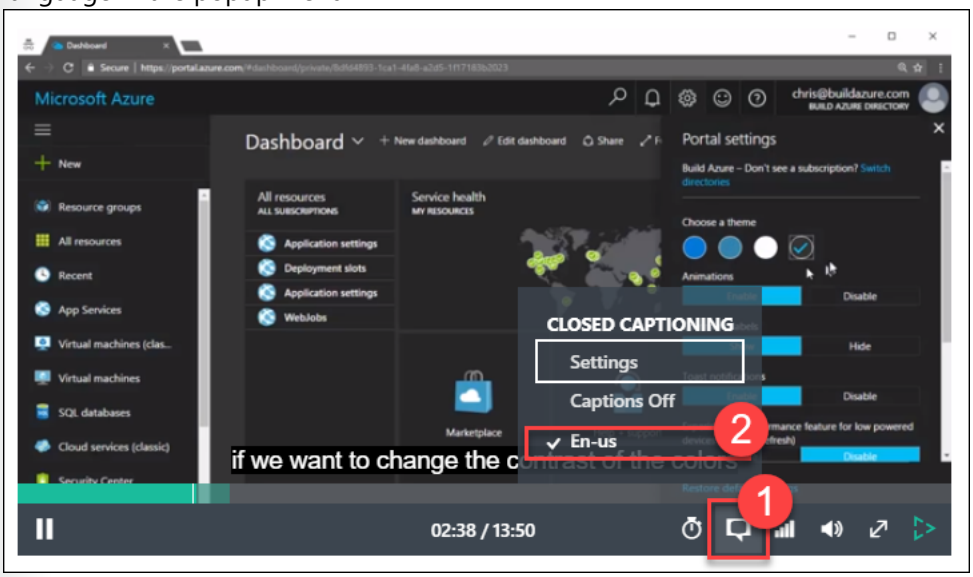


4. On the Video Player page, click the **Transcript** tab within the Video Insights. This will show you the Transcription of the Audio from the Video.

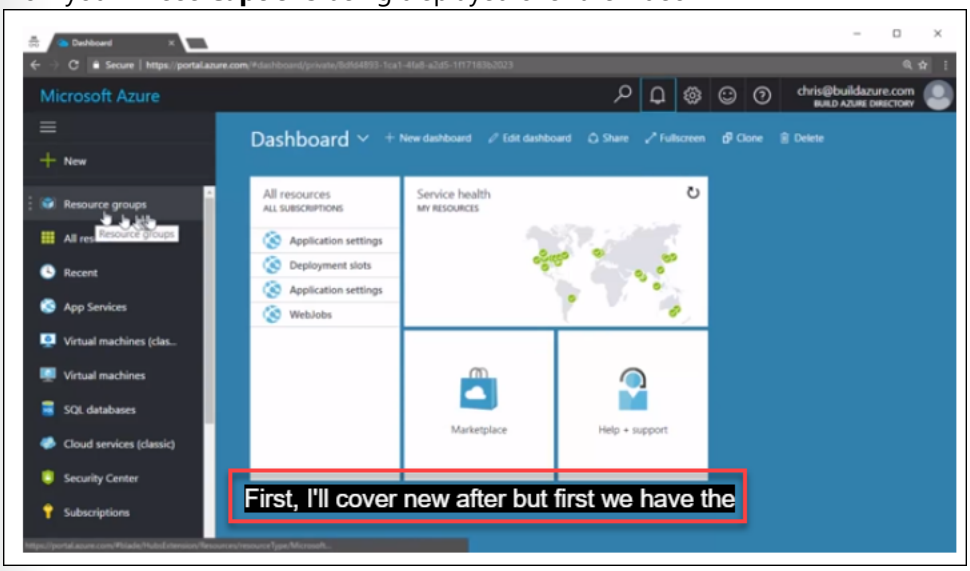



5. Notice as the video plays through, the Transcript highlights and automatically scrolls to show you the text you're hearing in the video

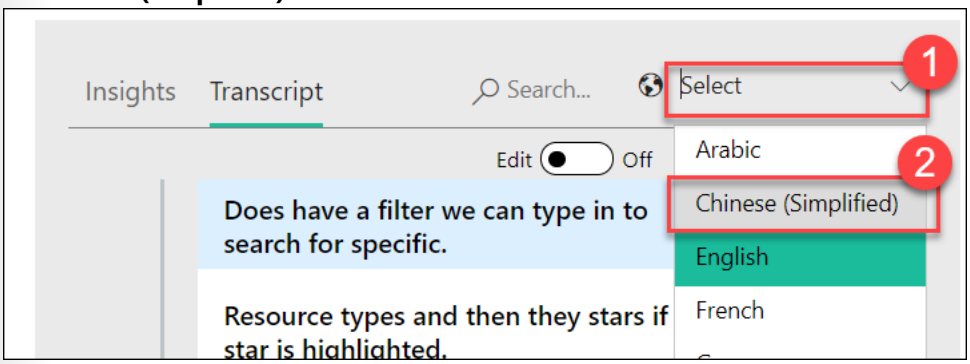
6. Hover over the **Video Player**, then hover over the **Closed Captions** icon, then click on the **En-us** (English) language in the popup menu



7. Now you will see **Captions** being displayed over the video




8. Click on the **Language** dropdown in the top-right of the **Video Insights** pane, then click to change the language to **Chinese (Simplified)**




9. The **Closed Captions** and the **Transcript** will now be automatically translated into **Chinese (Simplified)**

# After the hands-on lab

Duration: 10 minutes

## Task 1: Delete resources

1. Now that the hands-on lab is complete, go ahead and delete all the Resource Groups that were created for this lab. You will no longer need those resources and it will be beneficial to clean up your Azure Subscription.

You should follow all steps provided *after* attending the Hands-on lab.