Microsoft Power Apps and Azure

Hands-on Lab

February 2020

Table of Contents

Lab Overview and Objectives

Overview

Most enterprises have most of their core business data trapped in several silos (legacy databases and apps - Systems of Records). It is essential for digital transformation to break those data silos and make the information trapped in those systems available to employees and business teams as needed.

It is also important to modernize the user experience of the enterprise apps to drive productivity. Microsoft Power Apps along with Azure could be an effective way to bridge this gap, by enabling an RESTful API layer powered by Azure which could encapsulate and standardize the enterprise data for use with apps + Power Apps as the rapid application development user interface where business apps can be built with agility and in collaboration with the business.

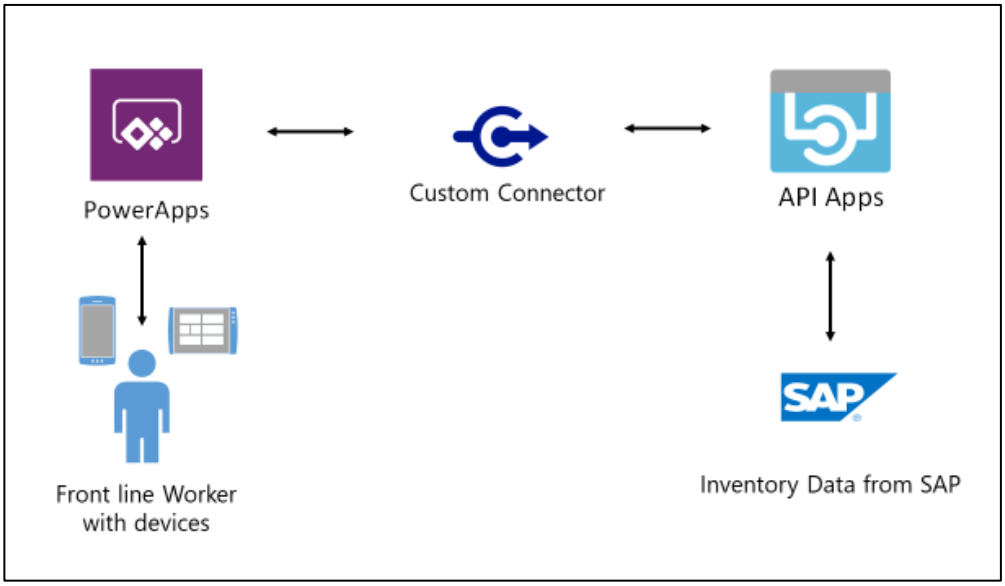
Objectives

This lab has two key objectives:

- To demonstrate how the Microsoft Power Apps platform can help unlock the potential of untapped assets within an Enterprise (legacy APIs, data sources, processes) with a low code / no code approach.

- Learn to create a cross-platform application user experience using Microsoft Power Apps to consume these RESTful APIs through custom connectors

Main components of the lab



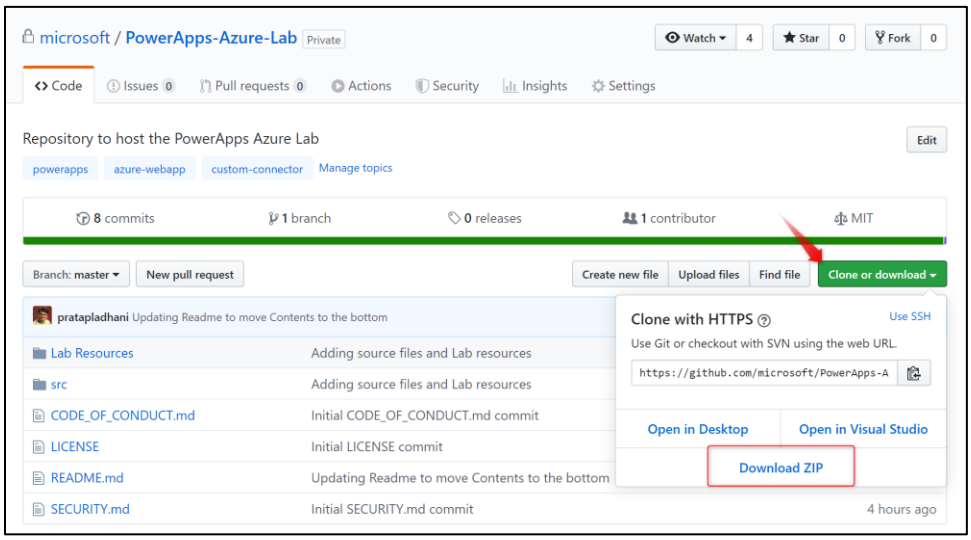
- **Microsoft Power Apps**: The end users consume the apps created from either a web-browser or through the Power Apps mobile apps from iOS or Android
- **Custom Connectors:** Allow the use of RESTful API operations within Microsoft Power Apps
- **API Apps:** An Azure service which allows you to build and host web apps, mobile back ends, and RESTful APIs

Exercise 1 – Build the API App

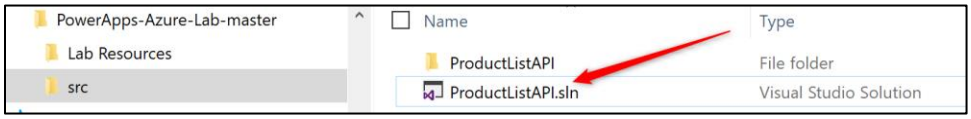
In this exercise, you will download a pre-built API app from a GitHub repo and test it locally before deploying it to Microsoft Azure

Task 1: Download from GitHub

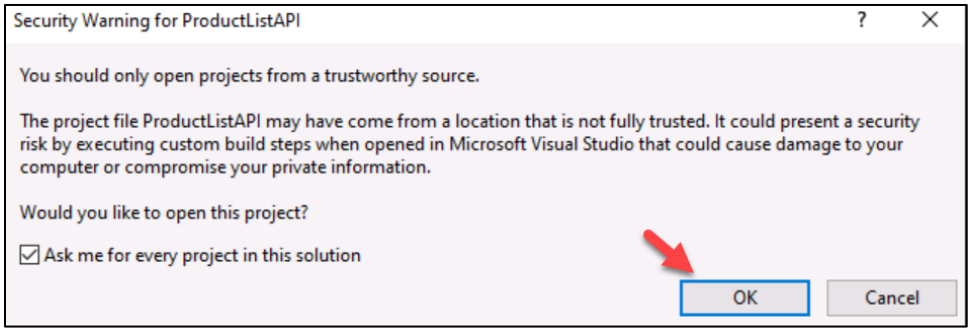
1) Navigate to https://github.com/microsoft/PowerApps-Azure-Lab
2) Click on the **Clone or download** and then **Download Zip**



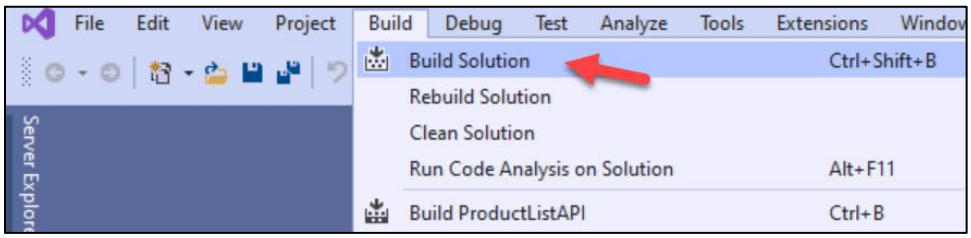
3) Extract the contents of the zip file and remember the location
4) In the extracted folder, open the **"src/ProductListAPI.sln"** Solution file in Visual Studio.



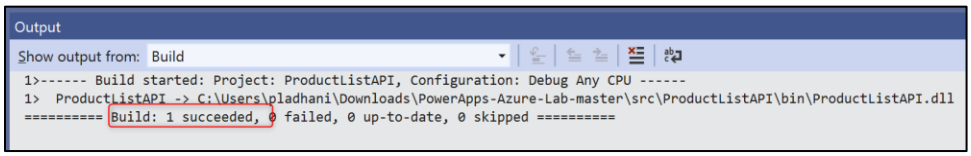
5) Select **Visual Studio 2019** and click OK
6) If you receive a security warning, click OK for the project as it loads



7) Build the Solution.

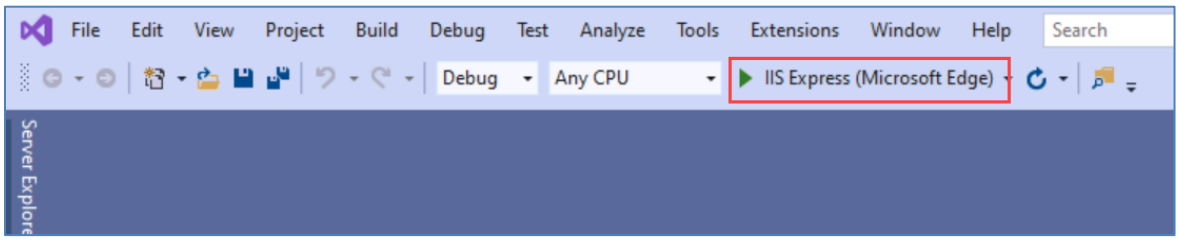


8) Confirm the success of the build and resolve any problems

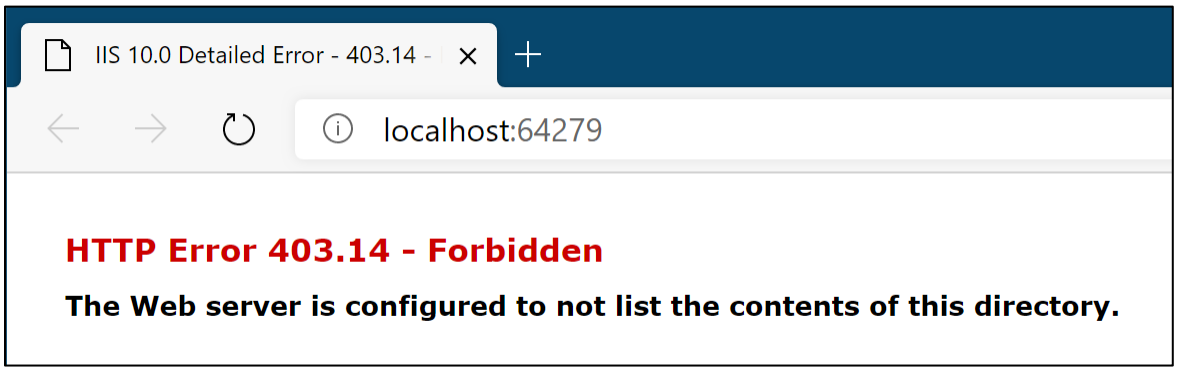


Task 2: Test the API

1) Click the IIS Express (Microsoft Edge) to launch the API app locally for testing

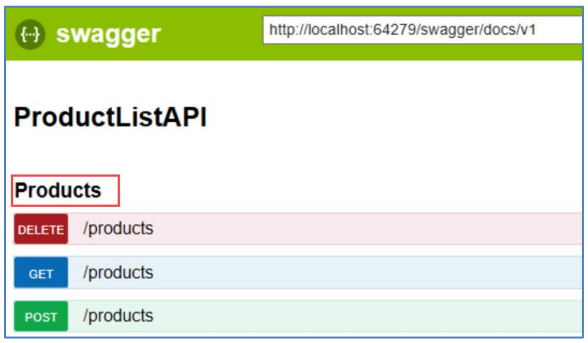


2) A new browser tab will open, and will initially show the following error, that is expected



3) Add **/swagger** to the end of the URL and then you should see the following which is the Swagger API Explorer.

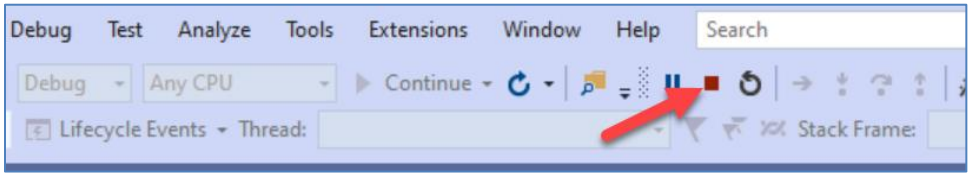
4) Click on **Products** to expand the list of actions available to test



5) Click on **GET** and it will expand and show you an example of what the API will return as a response. Click the **Try It out** to run the test

6) You should now see the response below the Try it out button and it will contain a list of all the products

7) Your API app is working locally, next we will deploy it to Microsoft Azure – Click the Stop button to end the debug session.



Exercise 2 – Deploy to Azure

In this exercise, you will deploy the API app to Microsoft Azure.

> ***Note:*** *You need an Azure Subscription for performing this step.*
>
> *If you don't have an Azure Subscription, you have the following options:*
> 1. *Use your monthly Azure credits included in various Visual Studio Subscriptions. Visit this link for more details:* https://azure.microsoft.com/en-us/pricing/member-offers/credit-for-visual-studio-subscribers/.
> 2. *Create a free Azure account from this URL:* https://azure.microsoft.com/en-us/free/.
> 3. *If your event team provided you a pre-provisioned resource-group or an Azure Pass – use that for the lab.*

Task 1: Configure the app service

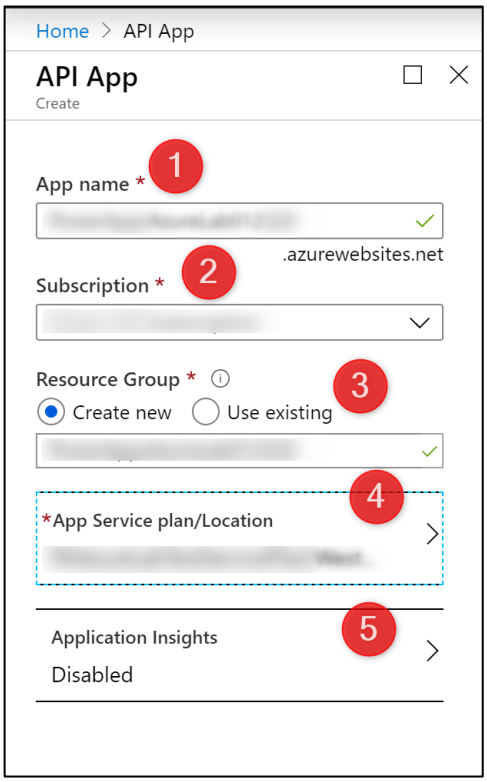
1) Create a new Web App in Microsoft Azure to host your API app, by visiting this URL: https://portal.azure.com/#create/Microsoft.ApiApp
2) Fill in the required information using the following
    1. **App Name:** Provide a unique name for your API App. This will be part of the URL for your API and will be followed by ".azurewebsites.net"
    2. **Subscription:** If you have multiple azure subscriptions, make sure you are using the right Azure Subscription.

       [If you are not seeing your subscription in the Subscription dropdown, chances are that you may be logged in on a different directory. You can switch the directory by selecting the Book and Filter icon on top of the Portal]

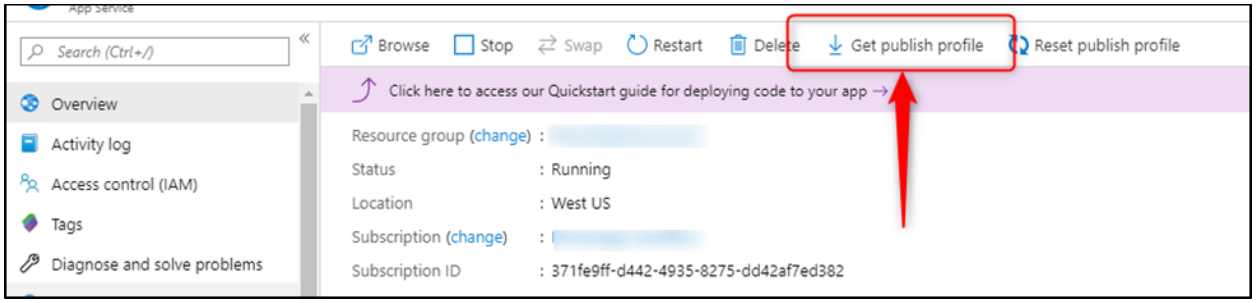
       

    3. **Resource Group:** Select an existing Resource Group or create new one
    4. **App Service Plan:** Choose an existing or create a new App Service Plan (choose the location of your preference and the Pricing tier of your preference)
    5. **Application Insights:** You can choose to keep it enabled or disable it if you want.

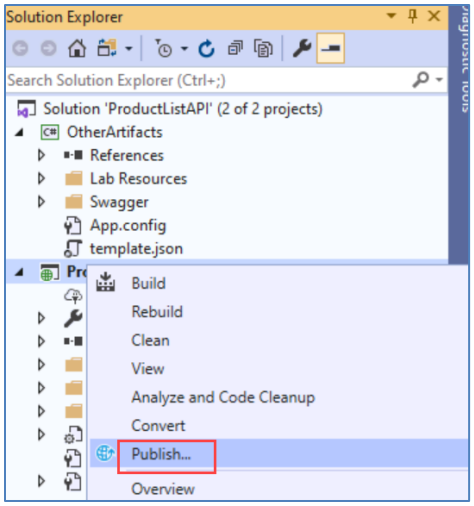
See the screenshot below for the options for creating the API App



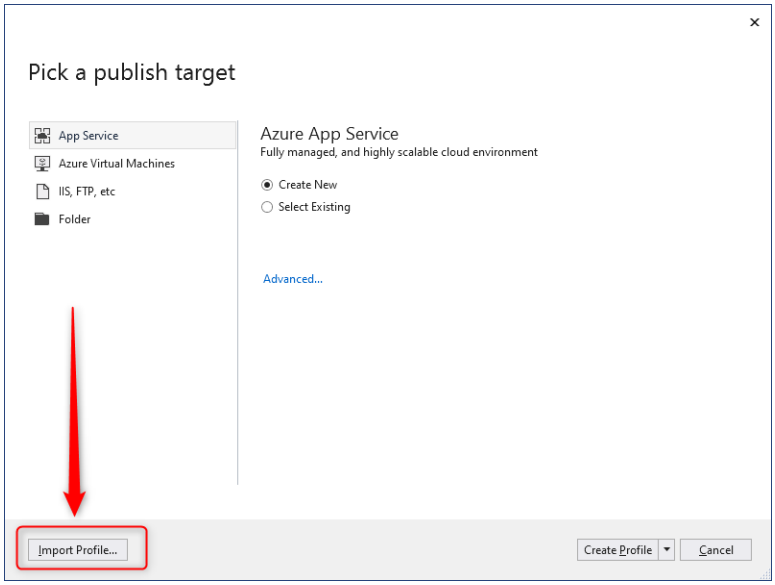
3) Wait for the deployment to complete and click the "Go to Resource" to browse to the newly created API App. Click on the "**Get publish profile**" icon on the top of the service and save it locally on your machine.



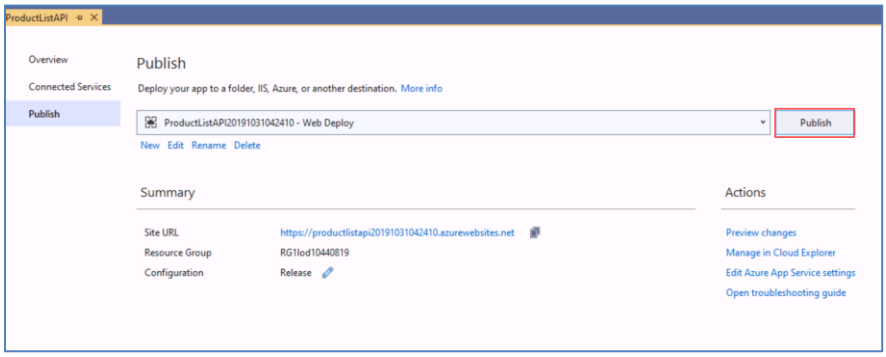
4) Back in Visual Studio - In Solution Explorer, right click on the **ProductListAPI** project and select **Publish**.



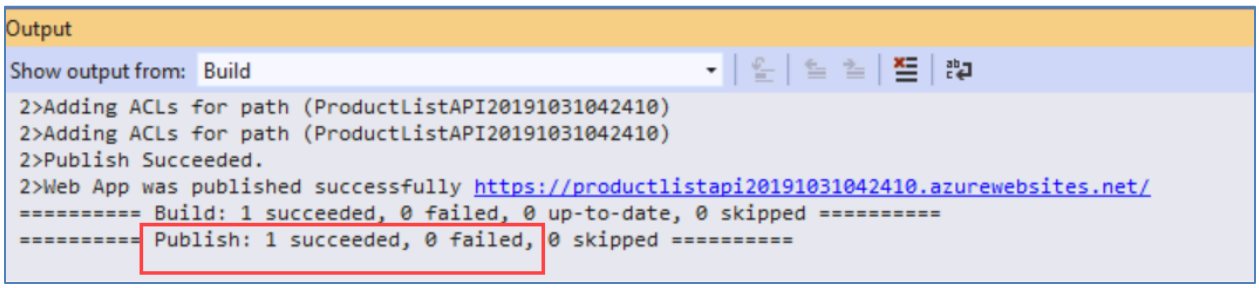
5) Select the **Import Profile** button and select the publish profile file that you had downloaded in the previous step



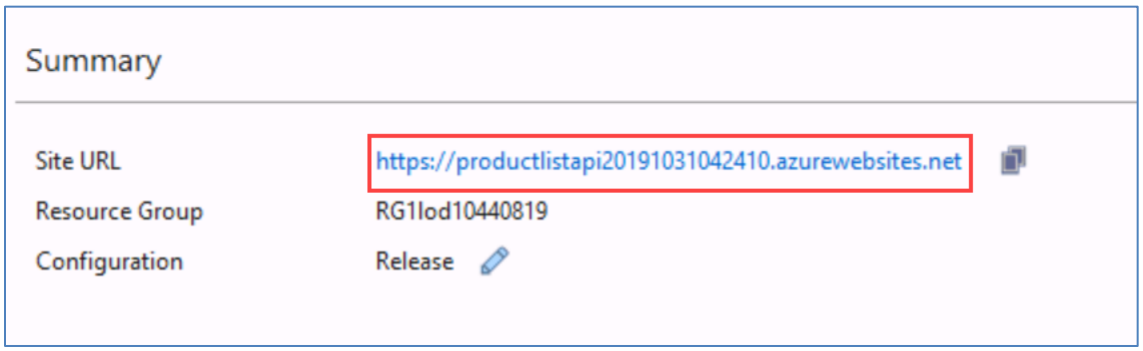
6) The App Service should now be ready to publish. Click the **Publish** button



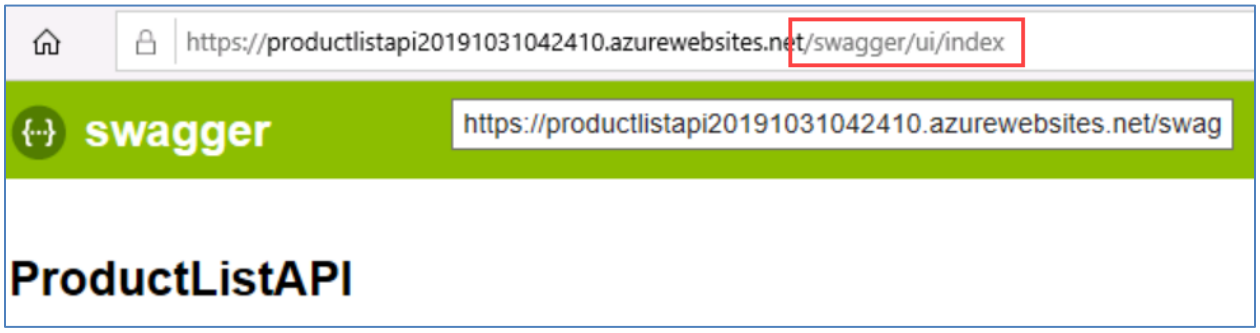
7) Confirm that the publish completed successfully



8) Click on the URL to your deployed site to launch it in the browser



9) Add **/swagger/ui/index** to the URL just like we did previously



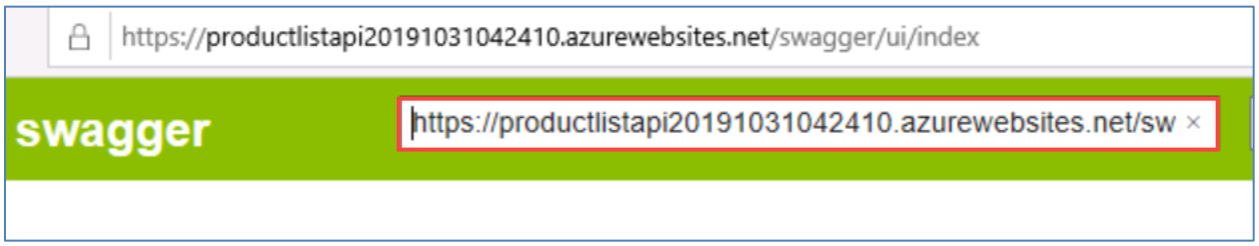
10) You now have the same API deployed to Azure. You can repeat the test we did previously on Exercise 1 : Task 2 running the GET on product list if you would like to double check it is working.
11) Leave the browser tab open as you will use it in the next exercise

Exercise 3 – Create the custom connector

In this exercise, you will create a Power Apps custom connector in Power Apps to use the APIs that we exposed in the previous exercise inside of a Canvas App. Read this article to learn more about Custom Connectors: https://docs.microsoft.com/en-us/connectors/custom-connectors/.

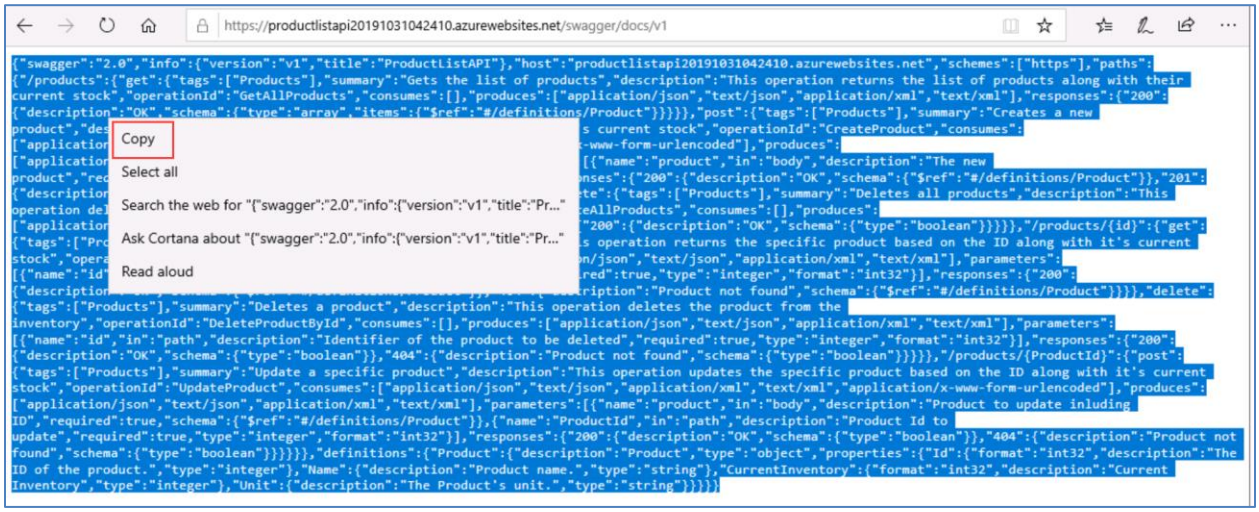
Task 1: Save the Open API definition

1) In the Swagger Explorer on your deployed site, copy the URL for the Open API definition. Make sure you copy the whole URL, some of it is hidden.

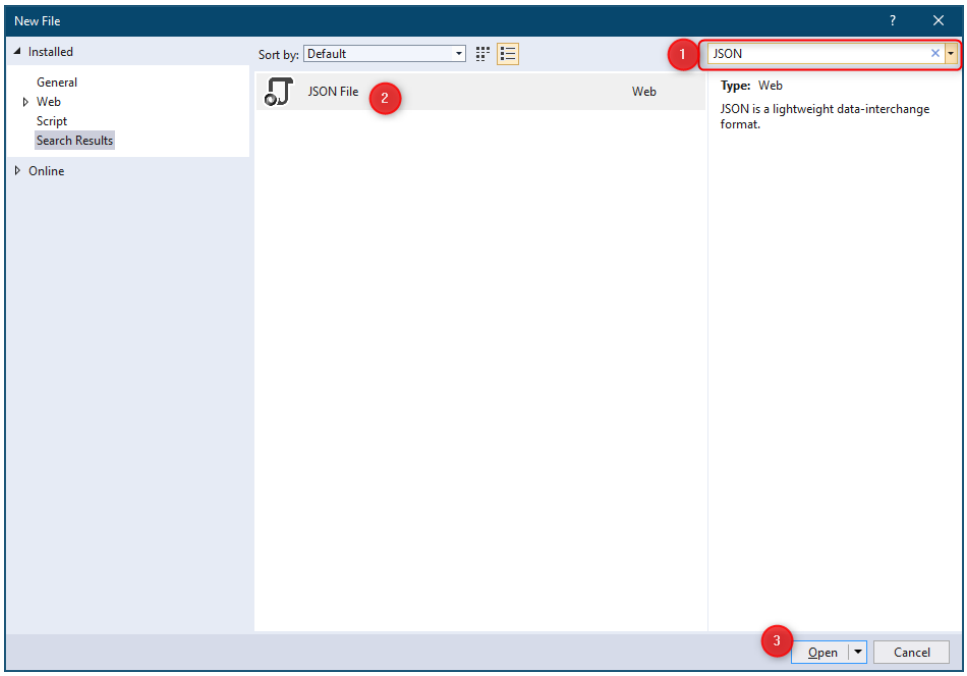


2) Paste that URL into the browser address bar and press enter. Your browser should load with the JSON which is the Open API/Swagger definition. We will need this saved to a file to import later.

3) Select all of the JSON and then right click and select Copy



4) Create a new JSON file in Visual Studio and press CTRL + A & CTRL+V to replace all the content with the copied JSON definition from the previous step. **Save** the file in the local Documents folder as **ProductOpenAPI.json**. Make sure to remember where you saved the file as you will need it in the next task..

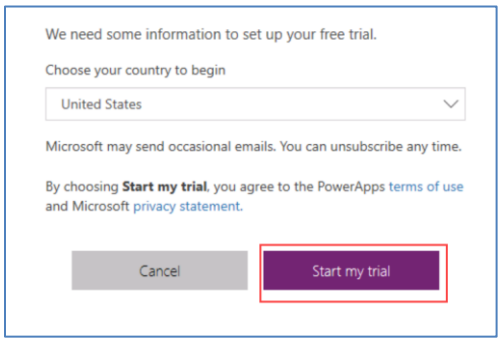


Task 2: Create the connector

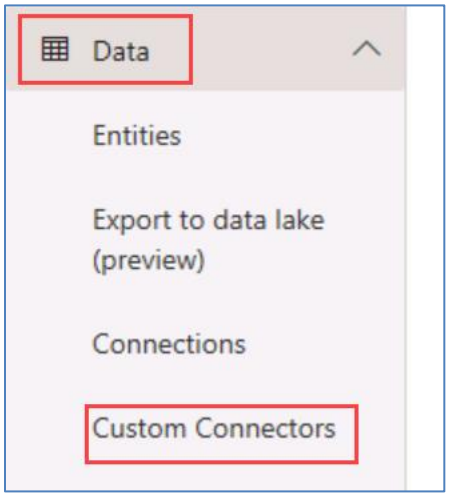
1) Navigate to https://make.powerapps.com
2) Log-in using the user credentials provided as part of your lab environment. Alternatively, you can also choose to use your own Organization credentials (Azure AD / Office 365 login).
3) [Optional] We recommend you sign up for the Power Apps Community Plan, which is a free development environment for individual use. Refer to this article for more details: https://docs.microsoft.com/en-us/powerapps/maker/dev-community-plan

   You can sign up for the community plan from the Power Apps Community Plan website. If you are an existing user of Power Apps with Office 365 or Dynamics 365, you can also create an environment for individual use.
   After signing up for the Community Plan, you will be redirected to the Power Apps site and will land in your individual environment. The environment is named with your name, for example 'John Doe's environment'. If there is already an environment with that name, the individual environment will be named as 'John Doe's (1) environment'.

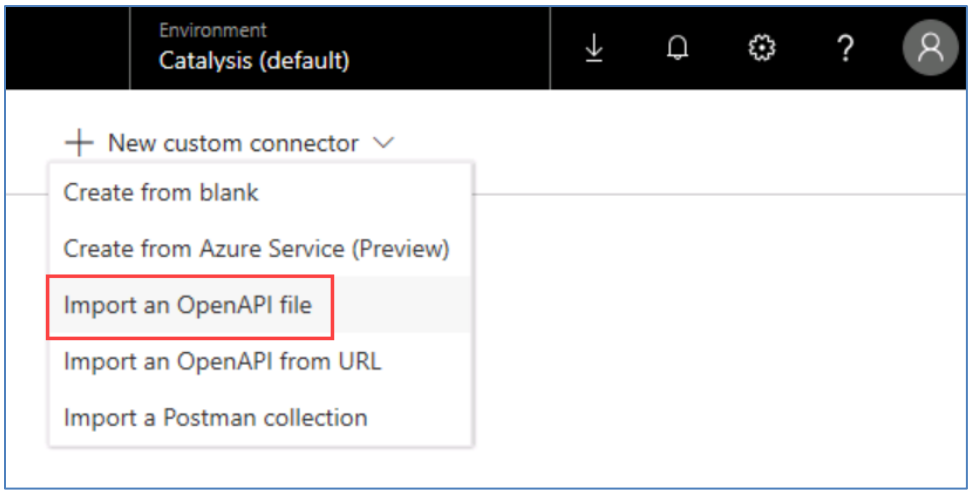
4) If you don't have a Power Apps License and aren't using the community plan, you might be prompted to start a trial – Click Start my trial to continue.



5) In the left navigation click **Data** to expand the menu and then click **Custom Connectors**

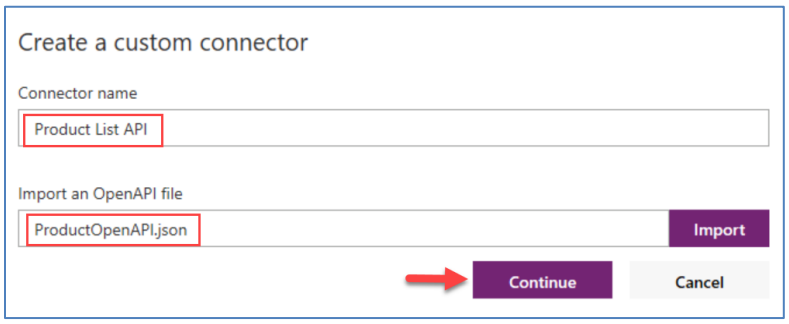


6) In the upper right corner click the **+ New custom connector** and select the **Import an OpenAPI file** option

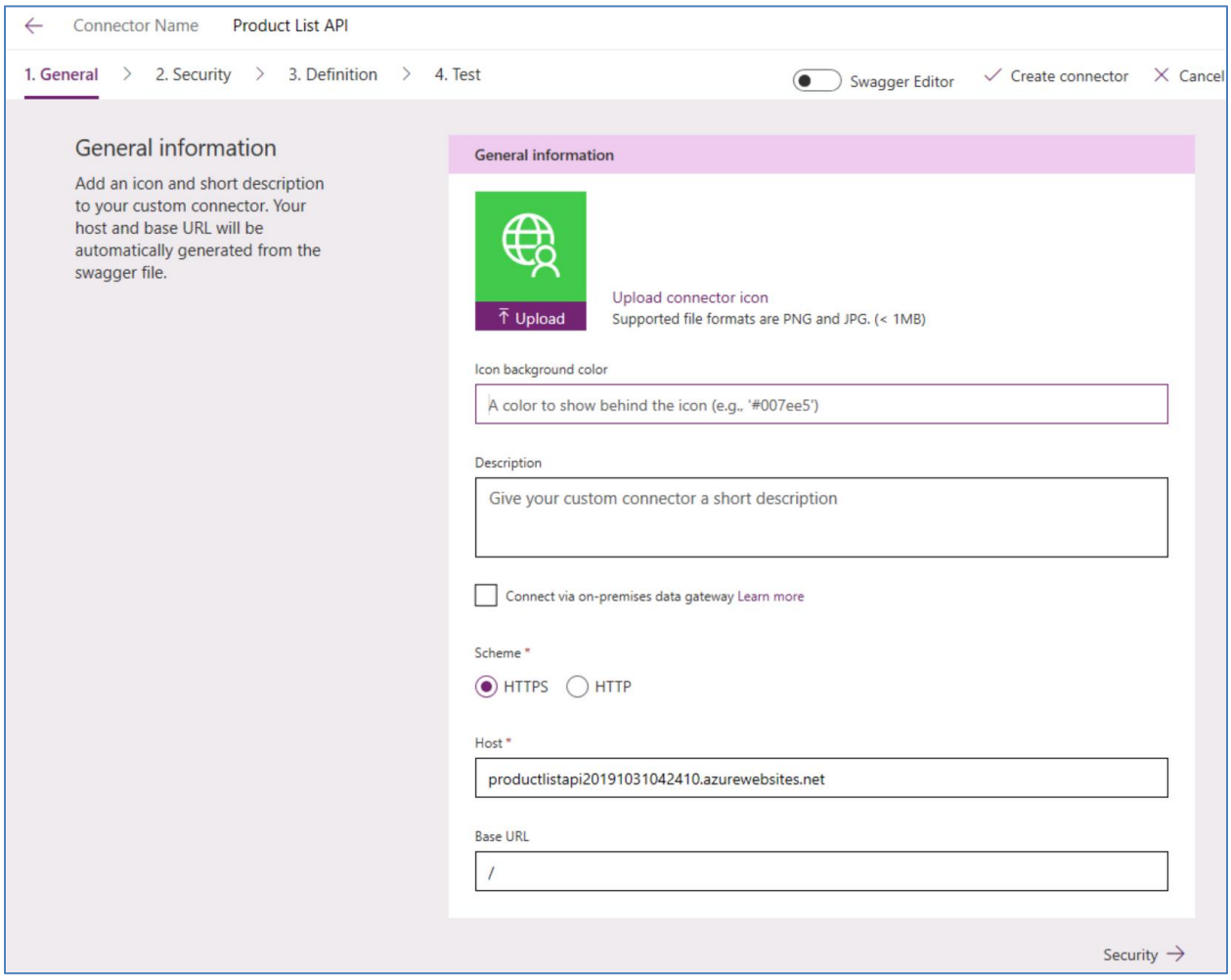


7) Input Product List API in the Connector Name field
8) In the Import an Open API File, click Import and browse to documents and select the ProductOpenAPI.json file you saved in the prior task

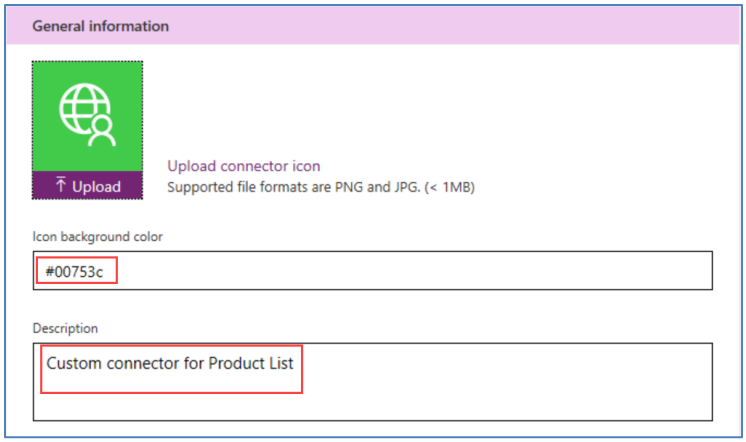
9) The information should look like the following and then click Continue



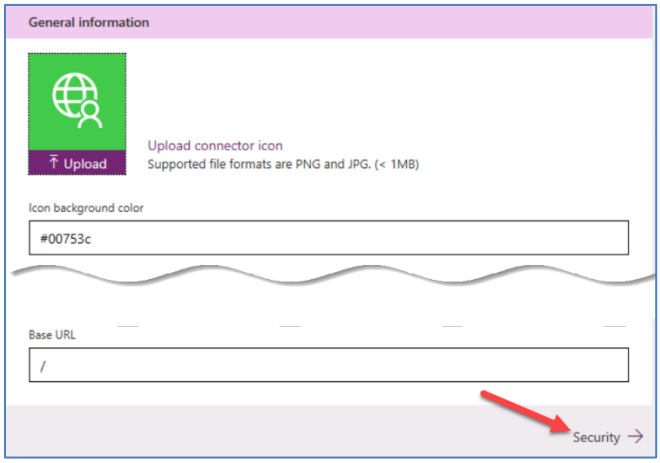
10) After the import completes the custom connector editor will load and should look like the following



11) Input #00753C in the Icon Background Color
12) Input Custom connector for Product List

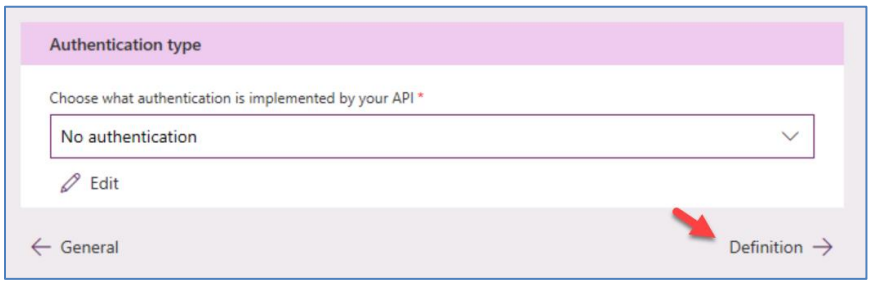


13) In the lower left corner of the General Information click Security to advance.

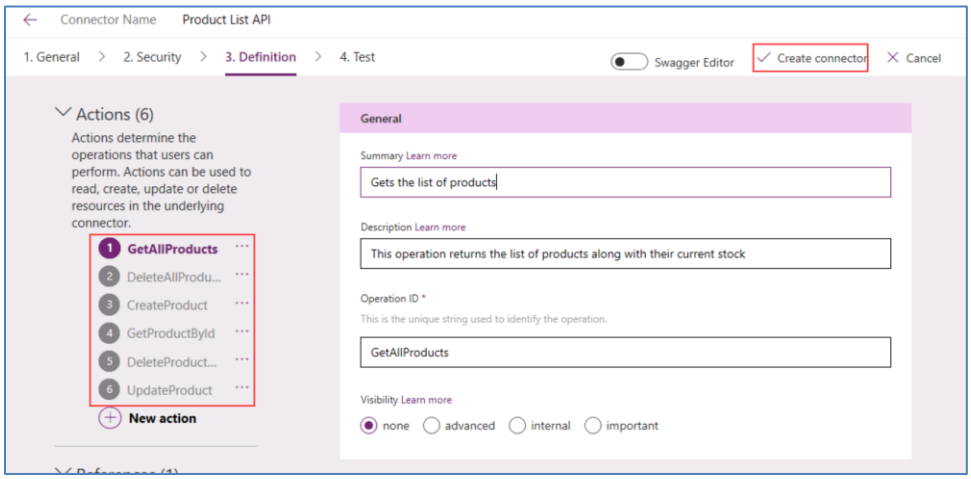


14) Review the list of authentication options, but leave it set at **No Authentication** and click **Definition** to advance.

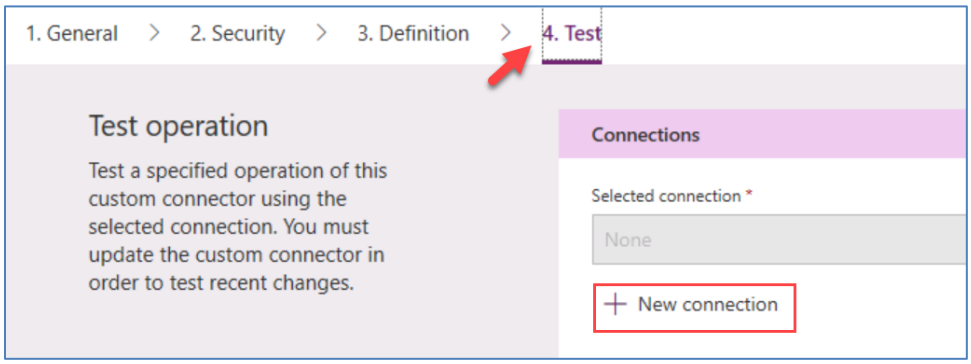
> ***Note:*** *This is not advisable to use "No authentication" in Production APIs – we are skipping this only for saving time in this lab.*
> *Refer to this article : https://docs.microsoft.com/en-us/connectors/custom-connectors/azure-active-directory-authentication for the guidance for protecting the API endpoint with Azure AD.*



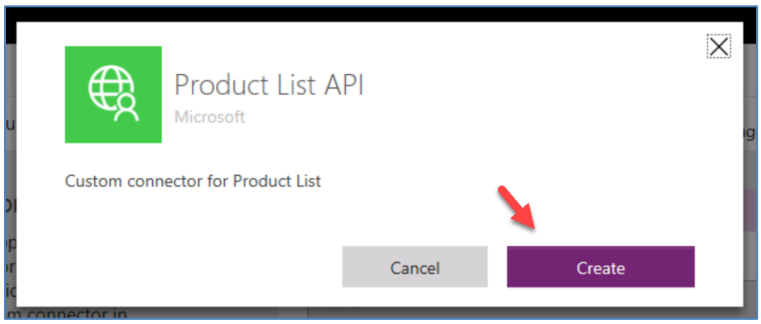
15) Review the connector definition, notice from the OpenAPI file imported it brought in 6 actions. After reviewing click **Create connector**



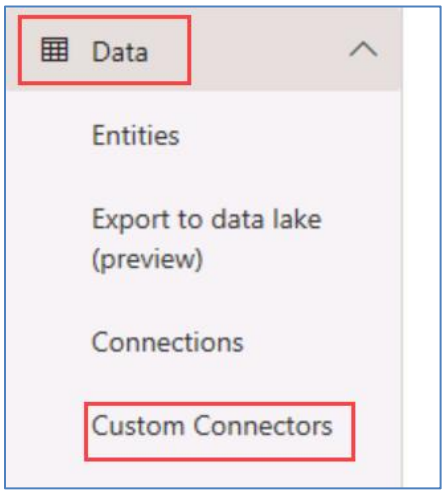
16) Click on the Test tab and then click +New Connection



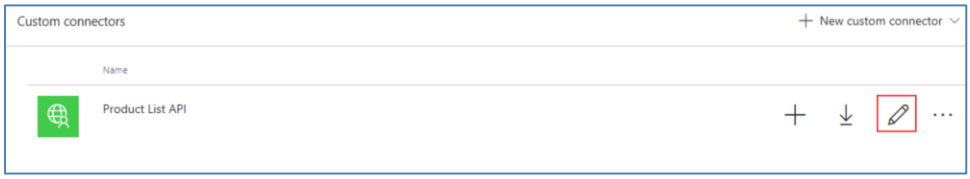
17) Click Create



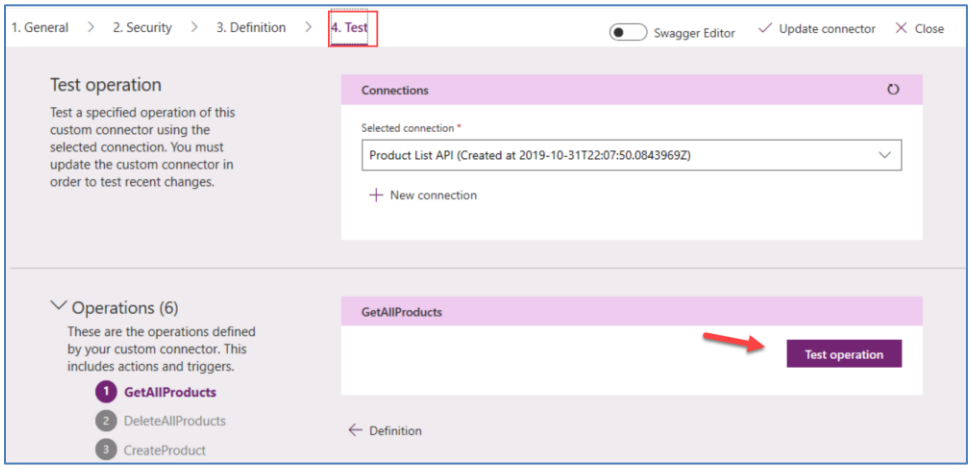
18) After clicking create, you will be navigated to your list of connections. Using the left navigation, navigate back to **Custom Connectors**



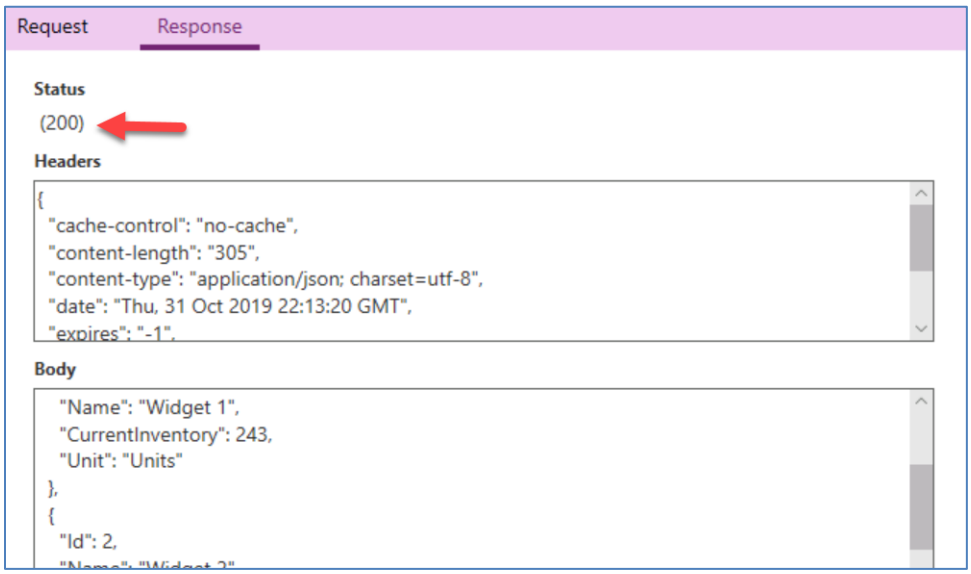
19) Select the custom connector you just created and click the edit icon



20) Change to the Test tab, and then click on the Test Operation for the **GetAllProducts** action



21) Confirm the Status is 200, and you should be able to review the Request and Response



22) You have successfully configured the custom connector.

Exercise 4 – Create the canvas app

Task 1: Create a Canvas App with Phone Layout

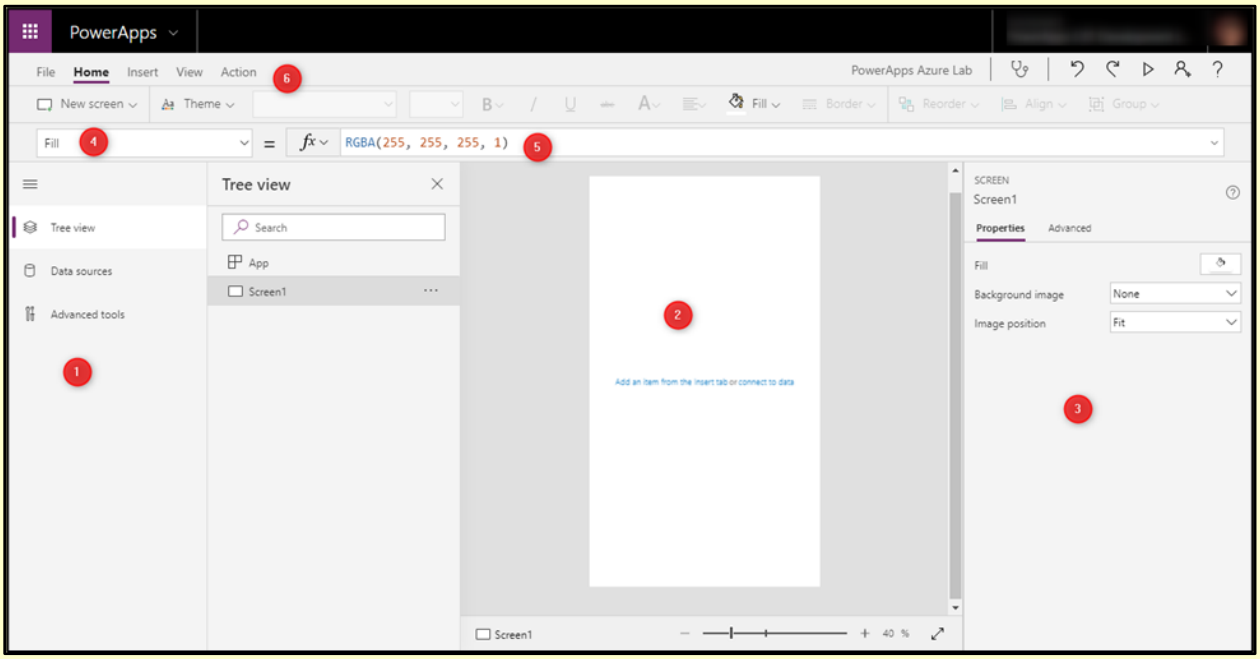
1. Navigate to *http://make.powerapps.com*

Power Apps Canvas Studio Layout (information only – no action required)

Power Apps Canvas Studio is available as a web application (http://make.powerapps.com) that you can use in any modern browser.

Power Apps Studio is designed to have a user interface familiar to users of the Office suite. It has three panes and a ribbon that make app creation feel **like building a slide deck in PowerPoint.** Formulas are entered within a function bar that is like Excel. Studio components:

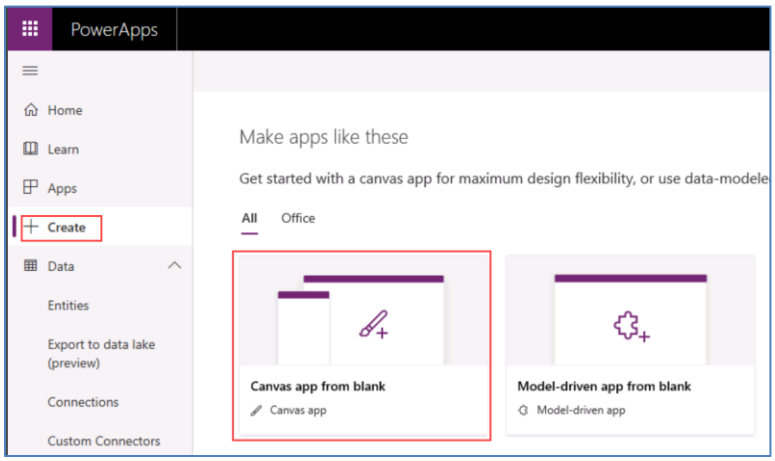
1. **Left navigation bar**, which shows the Tree view, Data sources & Advanced tools
2. **Middle pane**, which contains the app screen you are working on
3. **Right-hand pane**, where you configure properties for controls, bind to data, create rules, and set additional advanced settings
4. **Property** drop-down list, where you select the property for the selected control that you want to configure
5. **Formula bar**, where you add formulas (like in Excel) that define the behavior of a selected control
6. **Ribbon**, where you perform common actions including customizing design elements



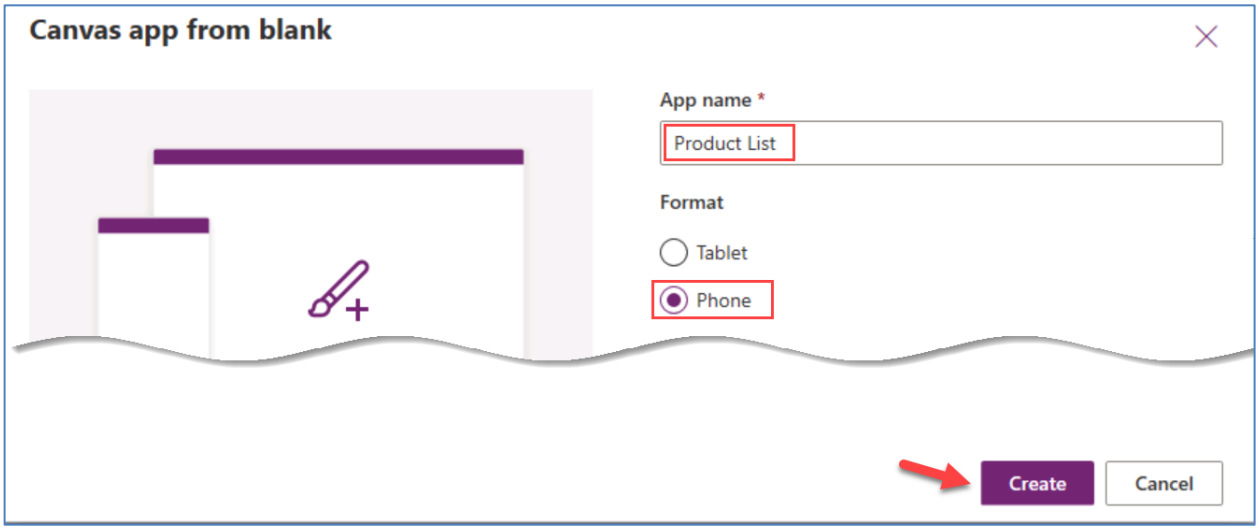
Locale-specific difference in formulas

Before you begin, please note that if the primary language set on your browser has its regional settings set to use the comma ',' for its decimal separator (like in much of Europe) your formulas will need to use a semicolon '**;**' instead of a comma '**,**' and 2 semicolons "**;;**" instead of a single semicolon "**;**" in your formulas. For example:	
en-US	`ClearCollect(colInventory, ProductListAPI.GetAllProducts());`
de-DE	`ClearCollect(colInventory; ProductListAPI.GetAllProducts());;`

2. Select + Create in the left navigation and choose the Canvas app from blank template

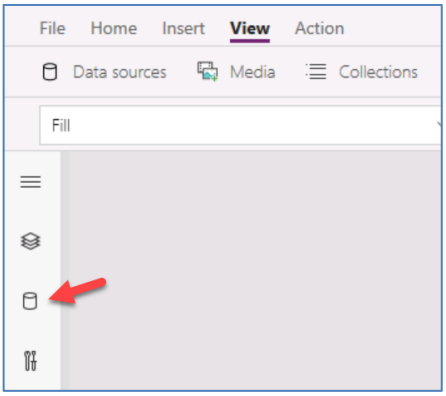


3. Input Product List as the App Name
4. Select Phone for the Format
5. Click Create to create the app

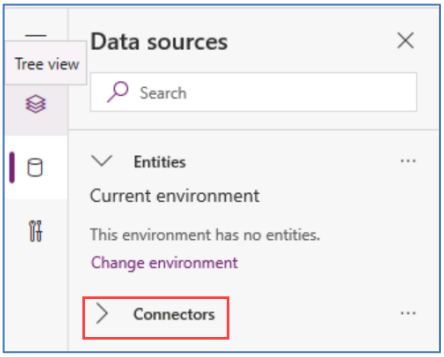


Task 2: Add your custom connector

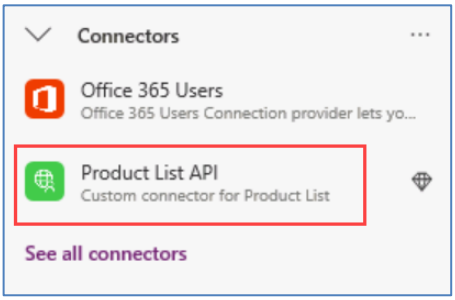
1. In the left tool bar click on the Data Source icon



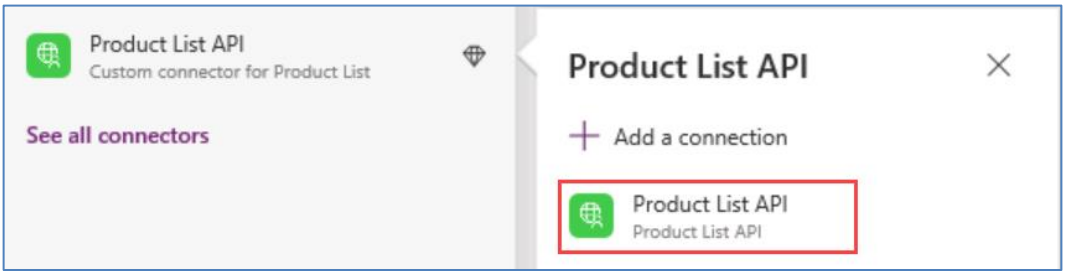
2. Click on > Connectors to expand the section



3. Click on the Product List API connector

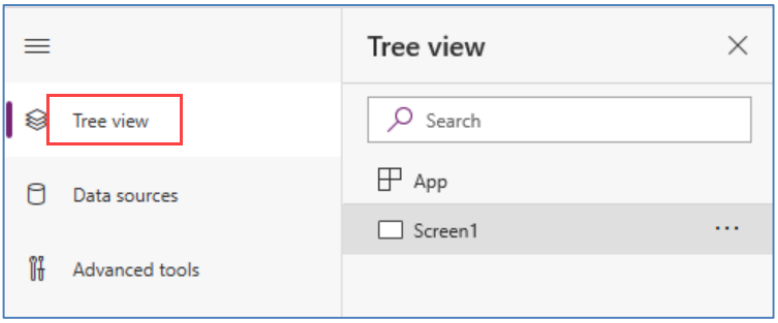


4. Then in the fly out click on Product List API again to associate the existing connection with your app

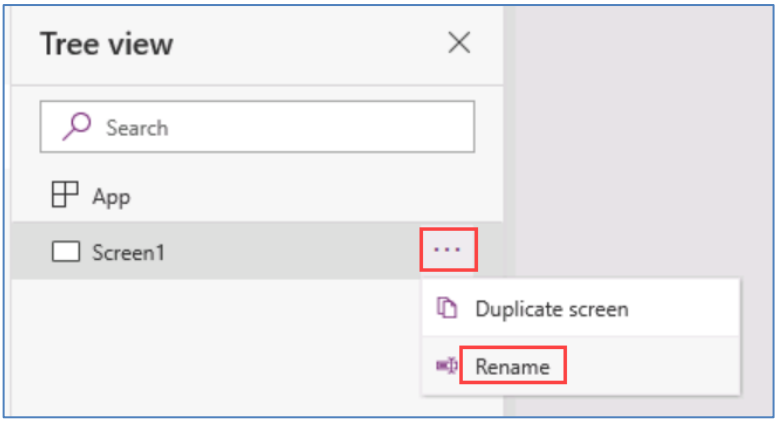


Task 3: Bind Data to the Gallery

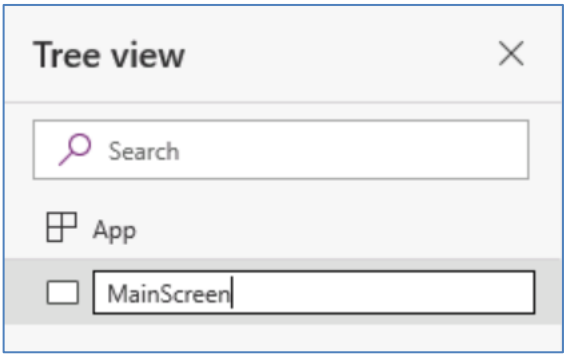
1. Change to Tree view to see the screens in the app



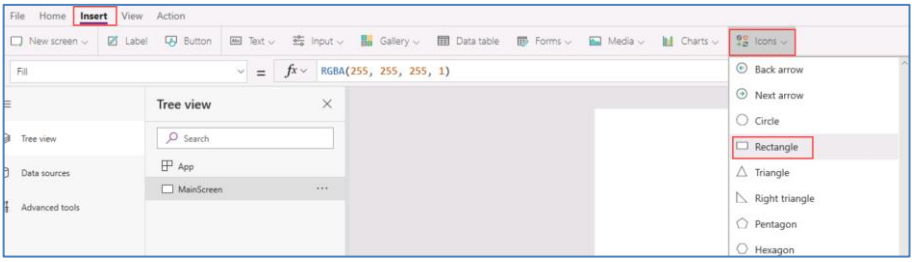
2. In the Tree view click … next to Screen1 and Select Rename



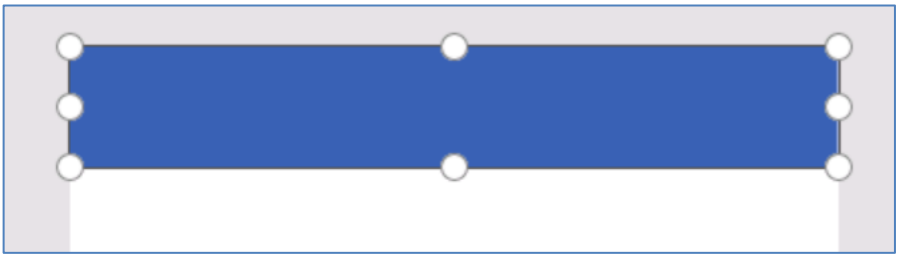
3. Rename Screen1 to MainScreen



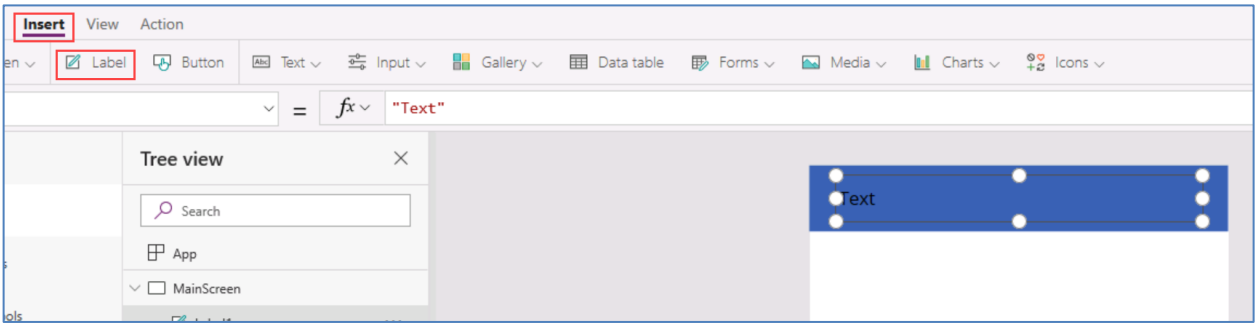
4. Next, we are going to add a header, select Insert -> Icons and choose Rectangle



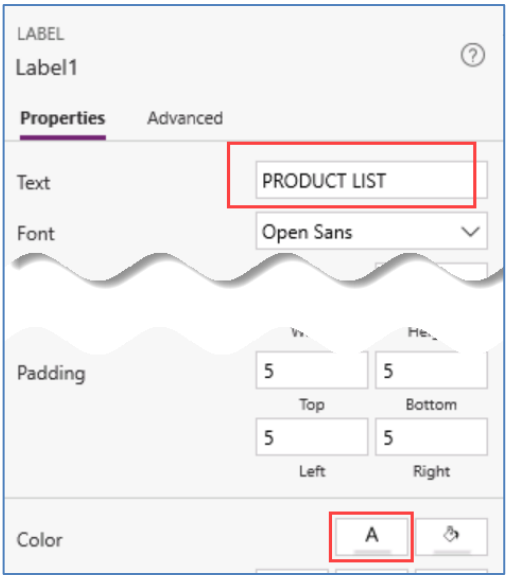
5. Drag the rectangle to the top of the screen and change the size to cover the whole top of the screen



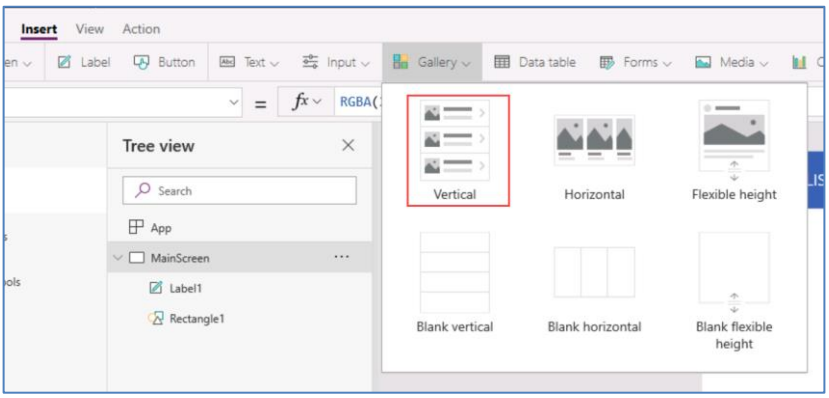
6. Select Insert -> Label and drag it up to be located in the header



7. With the label selected in the Properties panel on the right change Text to "PRODUCT LIST" and change Color to White



8. Insert a vertical Gallery Control



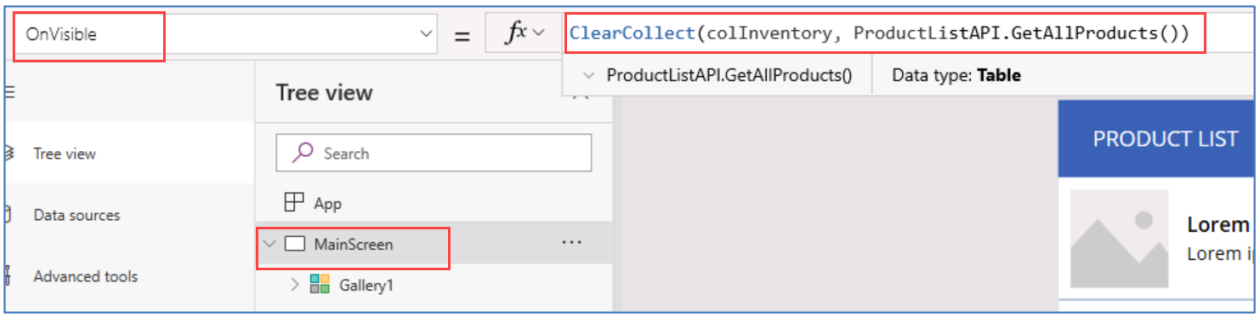
9. Move and adjust the size of the gallery you inserted to fit in the white space below the header.

10. Select **MainScreen** in the Tree View, In the property drop down select **OnVisible** and input the following formula. This action will call the GetAllProducts API on the OnVisible event of the Screen and save the response in a local collection called **colInventory**

```
ClearCollect(colInventory, ProductListAPI.GetAllProducts())
```

Use the following formula if you use comma "," as a decimal separator

```
ClearCollect(colInventory; ProductListAPI.GetAllProducts())
```

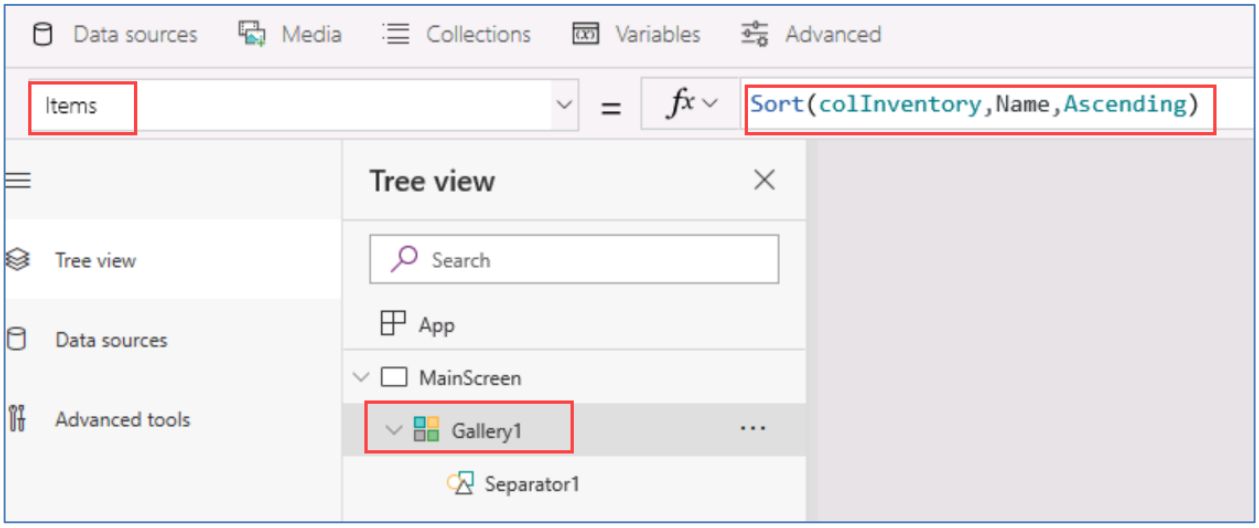


11. Select Gallery1=> Items property and type the following formula:

```
Sort(colInventory, Name, Ascending)
```

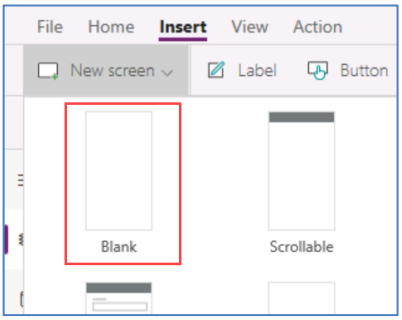
Use the following formula if you use comma "," as a decimal separator

```
Sort(colInventory; Name, Ascending)
```



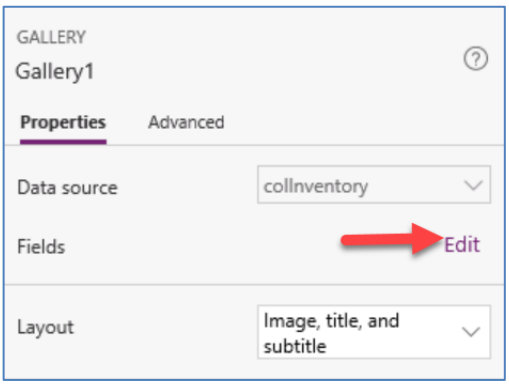
Note: The Gallery won't show data immediately, because the OnVisible event for the MainScreen would not have fired since we are already on this screen. Perform the next 2 steps where Screen2 is Visible and then when we open MainScreen again, it's OnVisible event shall fire and then you should be able to see the gallery populated with data.

12. Create Screen2 that we will use as the detail screen later by selecting **Insert** > **New Screen** and then the **Blank** screen template

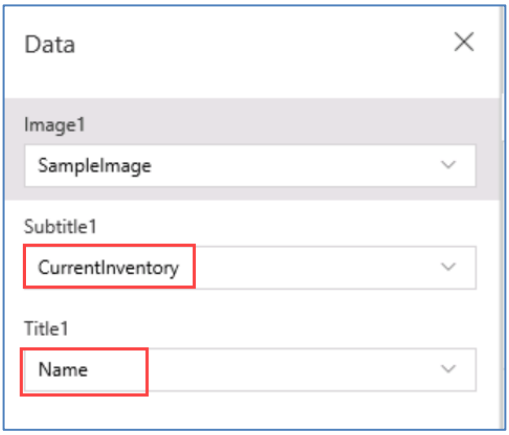


13. In the Tree View select MainScreen again. This will ensure the OnVisible function will be triggered for the MainScreen and the data is loaded to the collection. You should now see data from your API bound to your Gallery.

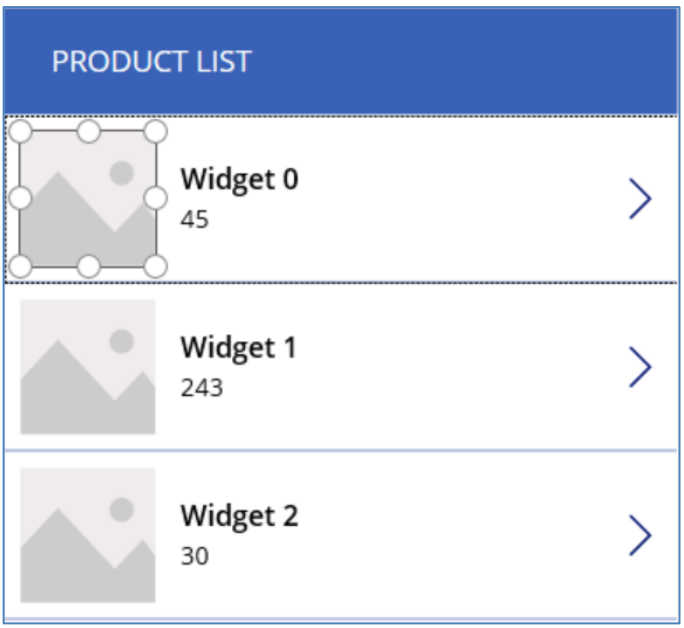
14. Select Gallery1 and click on Edit Fields in the right Property panel



15. In the Data panel, leave SampleImage for the Image1, select CurrentInventory for Subtitle1 and Name for Title1, and

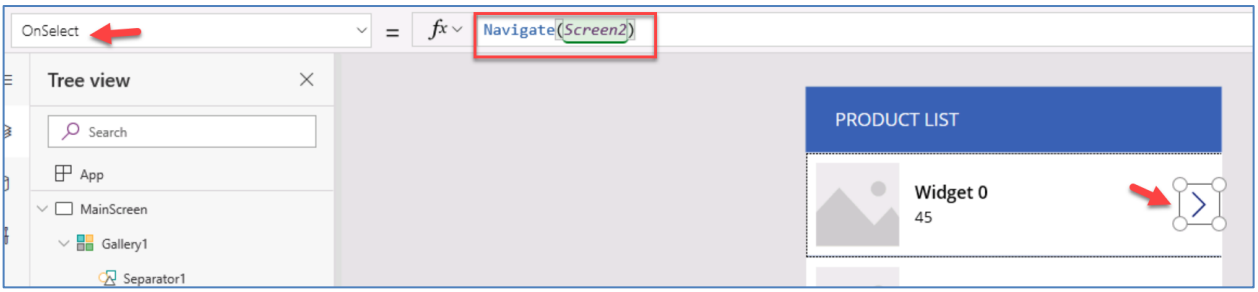


16. You should already start seeing data in the gallery and it should look something like the following image



17. Select the Detail icon on the first item in the gallery and input the following formula in the **OnSelect** property

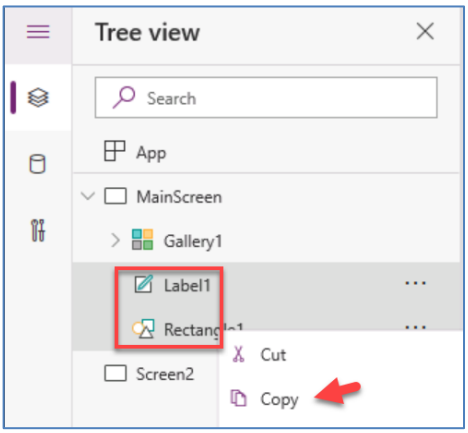
```
Navigate(Screen2)
```



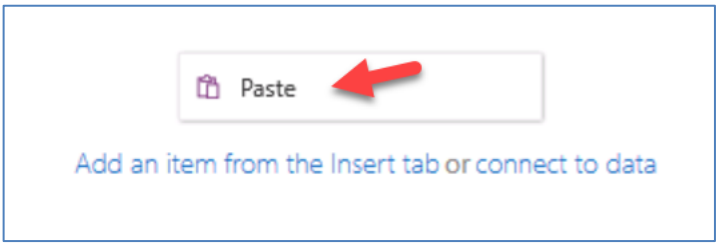
18. Save your app by clicking File -> Save and then clicking Save again in the lower right corner of the screen
19. Click the Back button to return to the app

Task 4: Create the Detail Screen to Edit and Delete Products

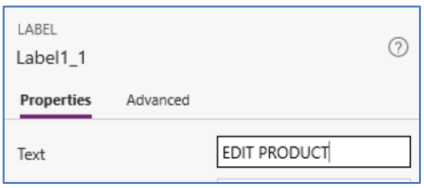
1. While holding the shift key, Select both Label1 and Rectangle1 from the Tree View inside the **MainScreen** group of controls and right-click Copy



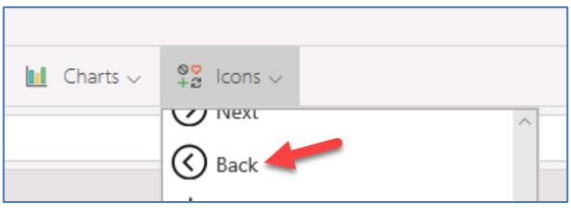
2. Click on Screen2 in the Tree View to make Screen2 the active screen
3. Right click on the screen surface and select Paste



4. While the controls are still selected drag them both to the top of the screen
5. Click on the Title label in the header and change the text in the right property panel to Edit Product



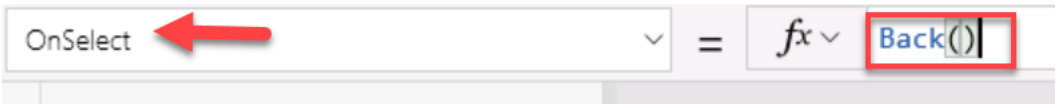
6. Insert a Back icon in the left side of the header



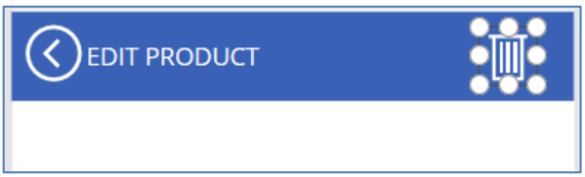
7. If necessary, drag the back icon into position on the left side of the header
8. With the back icon selected, in the right property panel change the Color to White

9. With the back icon still selected, in the property list choose OnSelect and in the formula type

```
Back()
```



10. Add Trash icon to the right side of the header to be a delete icon, change the color to white just like you did for the back icon

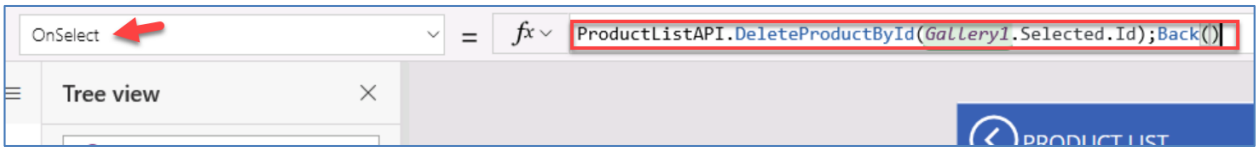


11. In the property selector above the Tree View choose **OnSelect** for icon2 and paste the following formula in the formula bar

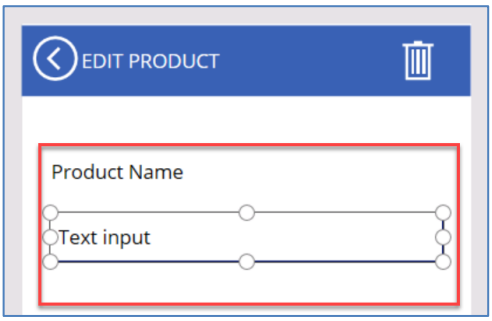
```
ProductListAPI.DeleteProductById(Gallery1.Selected.Id);Back()
```

Use the following formula if you use comma "," as a decimal separator

```
ProductListAPI.DeleteProductById(Gallery1.Selected.Id);;Back()
```

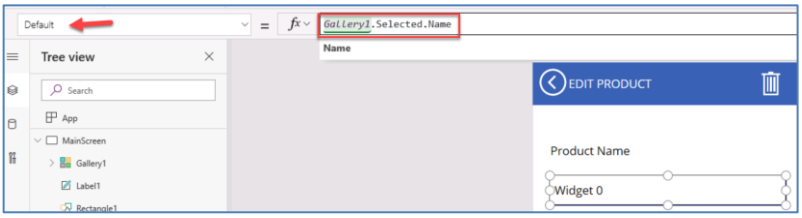


12. Insert a label control and change the text to be "**Product Name**"
13. Insert a **TextInput** control right below Product Name label



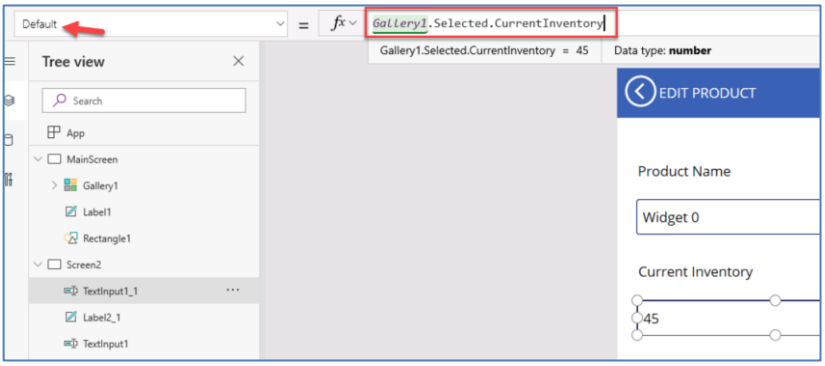
14. With the **TextInput** control still selected, set the behavior formula for the Default property as the following in the Formula Bar

```
Gallery1.Selected.Name
```



15. Below the last **TextInput** add a **Label** for "Current Inventory"
16. Add another **TextInput** control below the **label** and set the behavior formula for the Default property to:

```
Gallery1.Selected.CurrentInventory
```

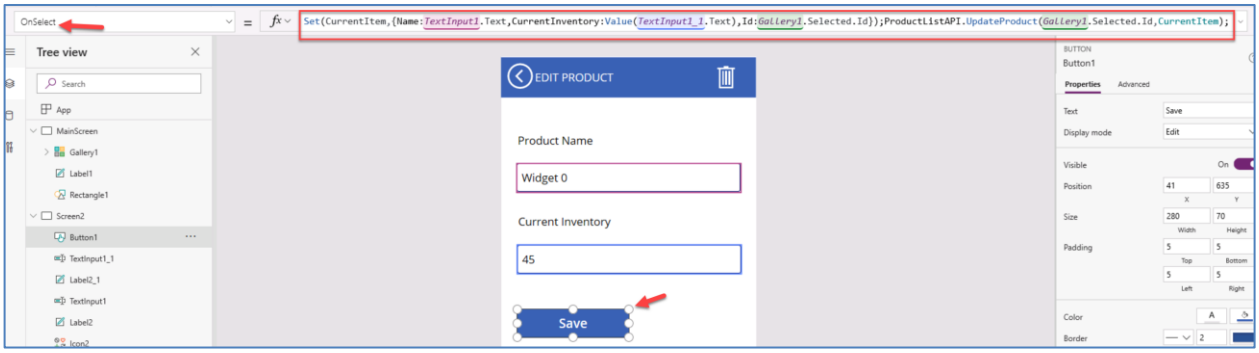


17. Insert a Button Control at the bottom of the screen, change the label to Save
18. Set the OnSelect property formula for the button to be the following formula. (**Note:** If you copied the first TextInput control and pasted it, it would be renamed as TextInput1_1, make sure to change the name of the control TextInput2 in the formula below to TextInput1_1 accordingly, in that case)

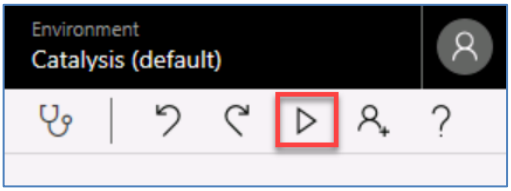
```
Set(CurrentItem, {Name:TextInput1.Text,
CurrentInventory:Value(TextInput2.Text), Id:Gallery1.Selected.Id});
ProductListAPI.UpdateProduct(Gallery1.Selected.Id, CurrentItem);
```

Use the following formula if you use comma "," as a decimal separator

```
Set(CurrentItem; {Name:TextInput1.Text;
CurrentInventory:Value(TextInput2.Text); Id:Gallery1.Selected.Id});;
ProductListAPI.UpdateProduct(Gallery1.Selected.Id; CurrentItem);;
```



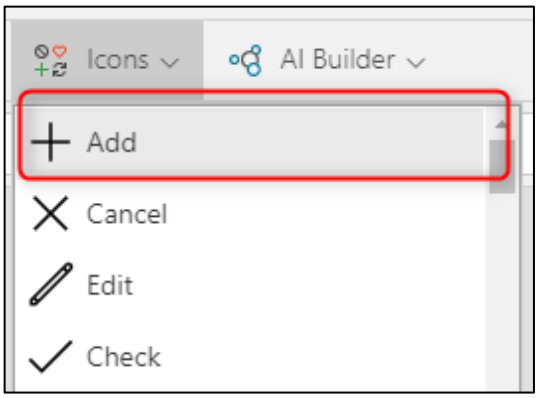
19. Save the app, using File -> Save
20. Click on the MainScreen in the Tree View to make it the active screen
21. Click the Play button to run the app in the upper right corner



22. Click on the Detail icon on one of the items to navigate to the detail page
23. Click the Delete button, you should return to the list with the item removed
24. Click on the Detail icon on another item
25. Change the values and click Save, then click the Back icon and review the revised value in the list

Task 5: Add functionality to Insert new Items

1. Select MainScreen and insert a Add "+" icon



2. If necessary, drag the Add icon into position on the right side of the header
3. With the Add icon selected, in the right property panel change the Color to White
4. With the Add icon still selected, in the property list choose OnSelect and in the formula type. We are setting the CurrentMode variable to "Add" and then navigating to the next screen.

```
Set(CurrentMode, "Add");Navigate(Screen2)
```

Use the following formula if you use comma "," as a decimal separator

```
Set(CurrentMode; "Add");;Navigate(Screen2)
```

5. Select the greater than icon in the first cell of the Gallery

6. Update the formula for OnSelect as the following which will set the CurrentMode variable to "Edit" and then Navigate to Screen2, when we select that icon

```
Set(CurrentMode,"Edit");Navigate(Screen2)
```

Use the following formula if you use comma "," as a decimal separator

```
Set(CurrentMode;"Edit");;Navigate(Screen2)
```

7. On Screen2, we don't to show the delete icon, if the CurrentMode is "Add". Select Screen2 and select the Trash icon. Select the Visible property and set it to this formula

```
CurrentMode<>"Add"
```

8. We also want to modify the title of the page to "ADD PRODUCT" and the button text to "Add" if the screen is in Add mode. We also don't want the textboxes to show any values from the gallery, if in Add mode.
9. Select the label of the title of the Screen2 and write this formula in the Text Property

```
If(CurrentMode="Add", "ADD", "EDIT") & " PRODUCT"
```

Use the following formula if you use comma "," as a decimal separator

```
If(CurrentMode="Add"; "ADD"; "EDIT") & " PRODUCT"
```

10. Select the Button on Screen2 and write this formula on the Text property

```
If(CurrentMode="Add", "Add", "Update")
```

Use the following formula if you use comma "," as a decimal separator

```
If(CurrentMode="Add"; "Add"; "Update")
```

11. Select the first InputText for the Product Name, change the formula on the Default property to the following

```
If(CurrentMode="Add", "", Gallery1.Selected.Name)
```

Use the following formula if you use comma "," as a decimal separator

```
If(CurrentMode="Add"; ""; Gallery1.Selected.Name)
```

12. Same way, select the second InputText for the Current Inventory, change the formula on the Default property to the following

```
If(CurrentMode="Add", "", Gallery1.Selected.CurrentInventory)
```

Use the following formula if you use comma "," as a decimal separator

```
If(CurrentMode="Add"; ""; Gallery1.Selected.CurrentInventory)
```

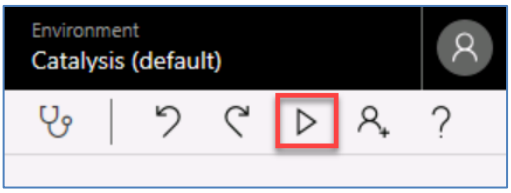
13. Also, we want to call the CreateProduct API instead of the UpdateProduct API if we are in Add mode. While the Button is selected, update the formula to the following for the OnSelect property

```
Set(CurrentItem, {Name:TextInput1.Text,
CurrentInventory:Value(TextInput1_1.Text), Id:Gallery1.Selected.Id});
If(CurrentMode="Add", ProductListAPI.CreateProduct(CurrentItem),
ProductListAPI.UpdateProduct(Gallery1.Selected.Id, CurrentItem));
```

Use the following formula if you use comma "," as a decimal separator

```
Set(CurrentItem, {Name:TextInput1.Text;
CurrentInventory:Value(TextInput1_1.Text); Id:Gallery1.Selected.Id});;
If(CurrentMode="Add"; ProductListAPI.CreateProduct(CurrentItem);
ProductListAPI.UpdateProduct(Gallery1.Selected.Id; CurrentItem));;
```

14. Click on the MainScreen in the Tree View to make it the active screen
15. Click the Play button to run the app in the upper right corner



16. Click on the Add icon on the top to navigate to the add page
17. Click the Delete button, you should return to the list with the item removed
18. Click on the Detail icon on another item
19. Change the values and click Save, then click the Back icon and review the revised value in the list

Task 6: [Optional] Run the same app on your phone (iOS or Android)

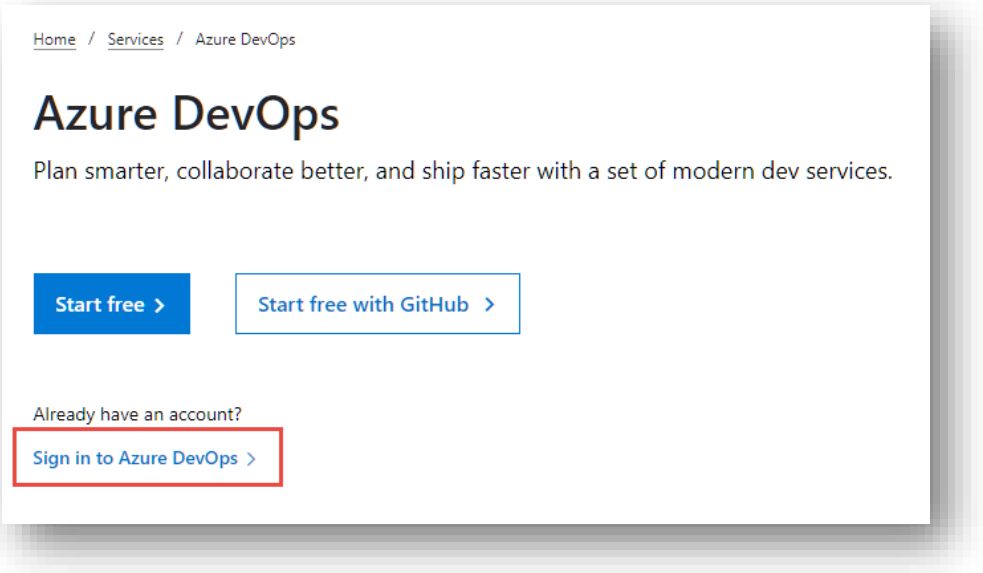
1. Download the Power Apps client app from the iOS or Android store or from the links below
    - iOS: https://aka.ms/powerappsios
    - Android: https://aka.ms/powerappsandroid
2. Login on the app using the same credentials used for this lab and run the app

Bonus Lab 5: [Optional] Enable the app to support Application Lifecycle Management (ALM) and setup Azure DevOps

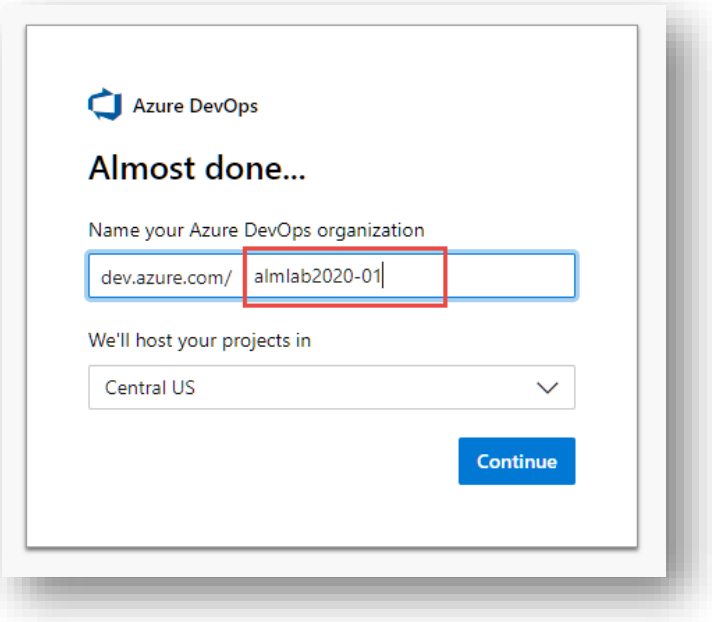
Contents

Set up Azure DevOps Subscription

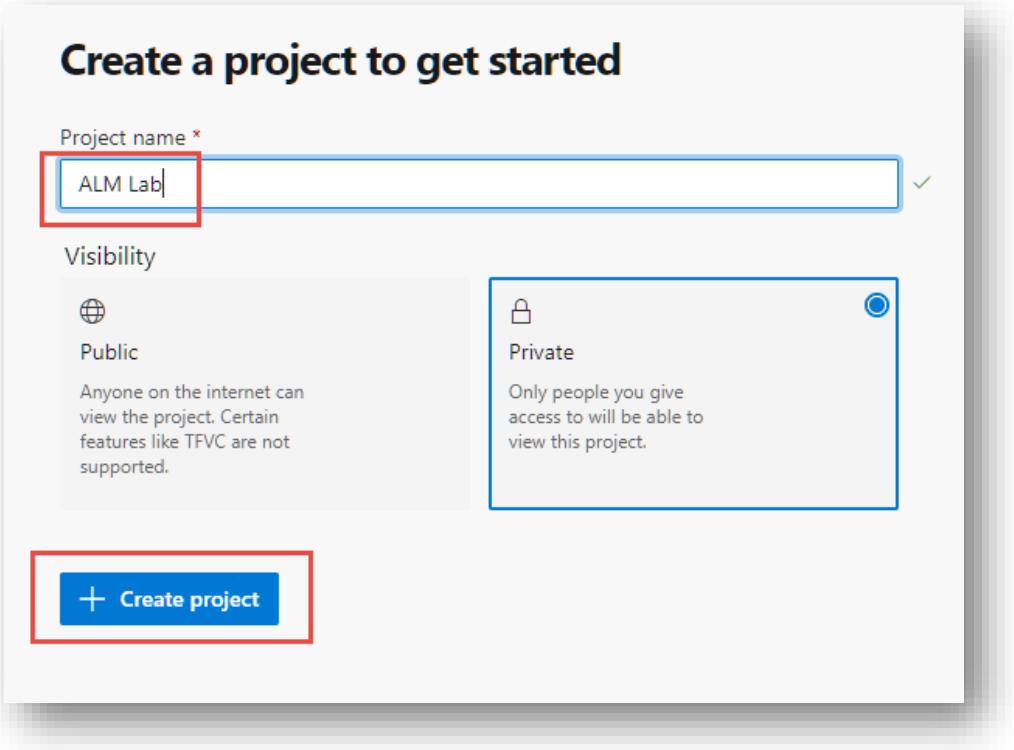
1. Visit https://dev.azure.com/ and click on the Sign In to Azure DevOps link (NOTE: Use a private/incognito browsing session to setup your own organization if you do not already have one)



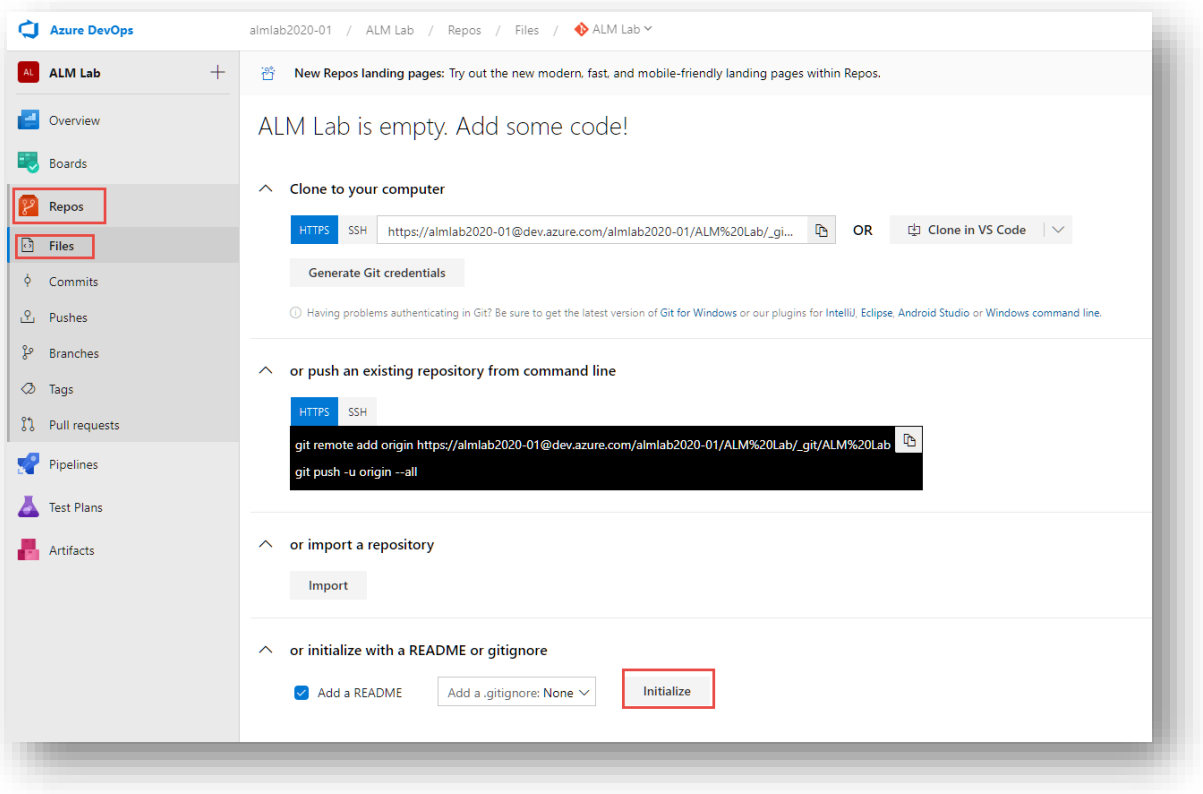
2. Go through the initial setup screens until you get prompted to name your Azure DevOps organization. Create a unique name and save the link for easy access later



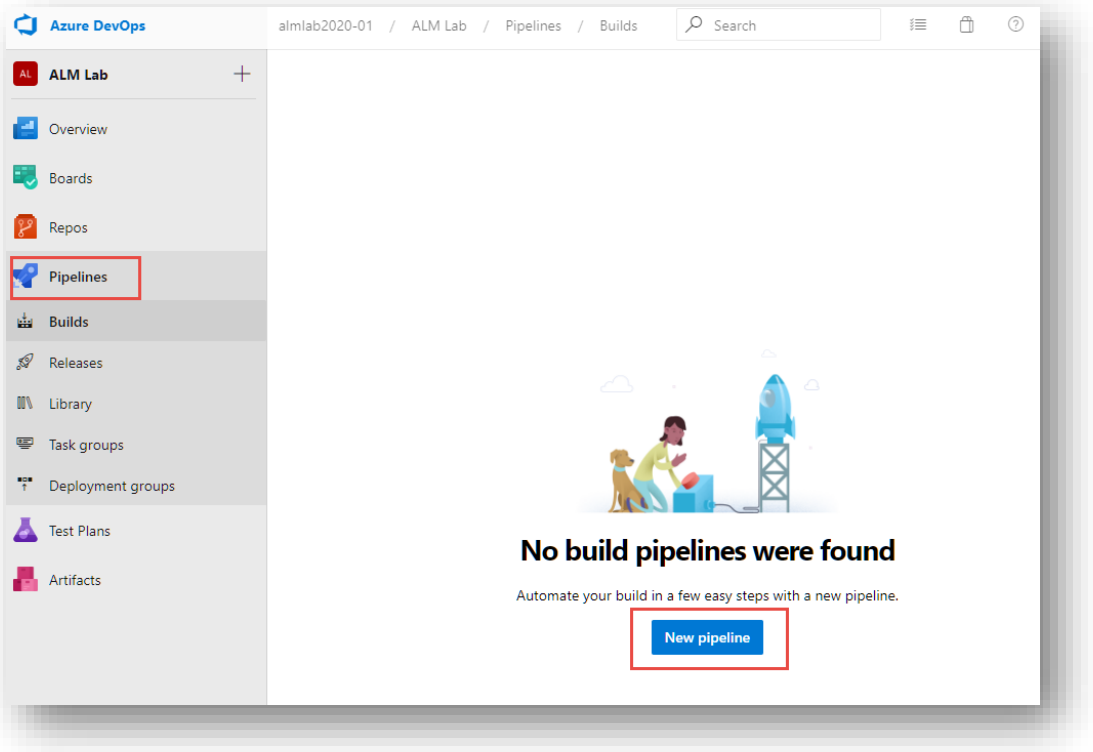
3. After the organization has been created, enter a project name and click the Create Project button



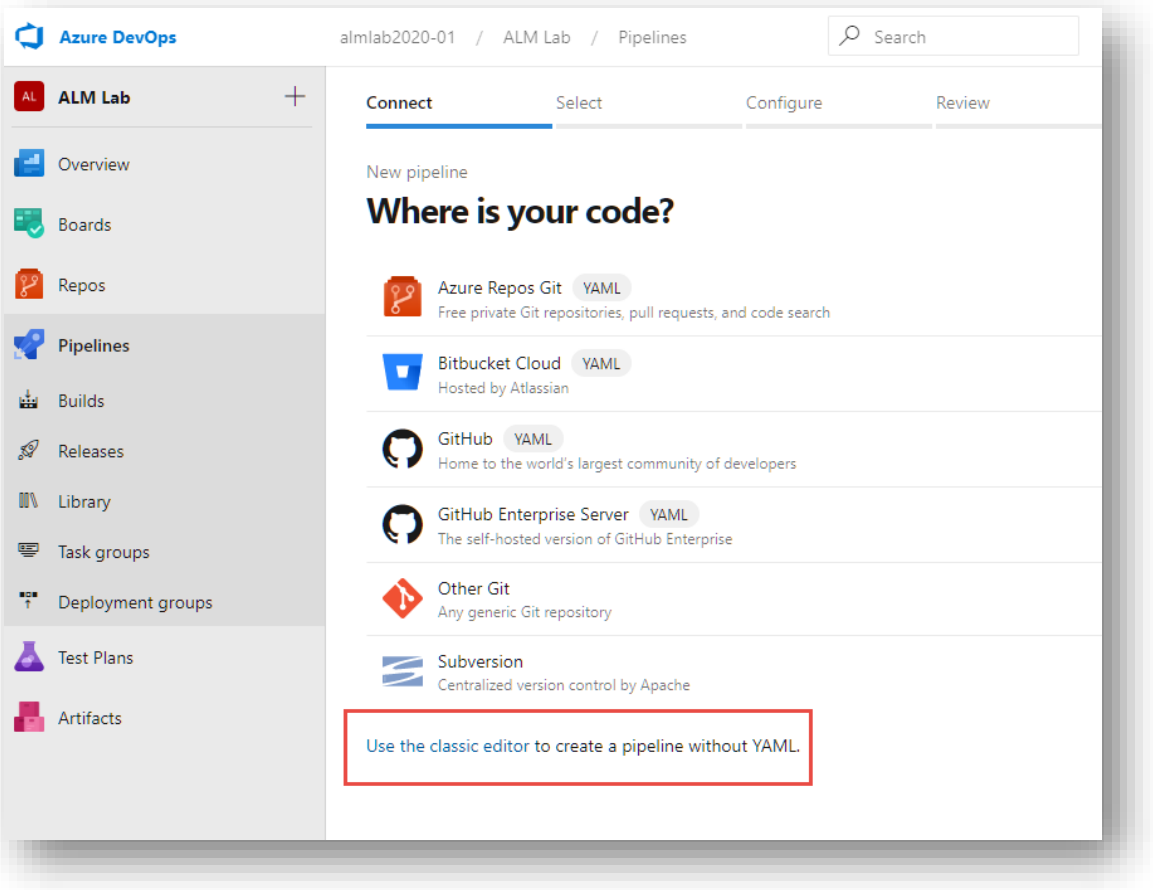
4. Initialize your repo by clicking Repos, then Files, and Initialize with readme.



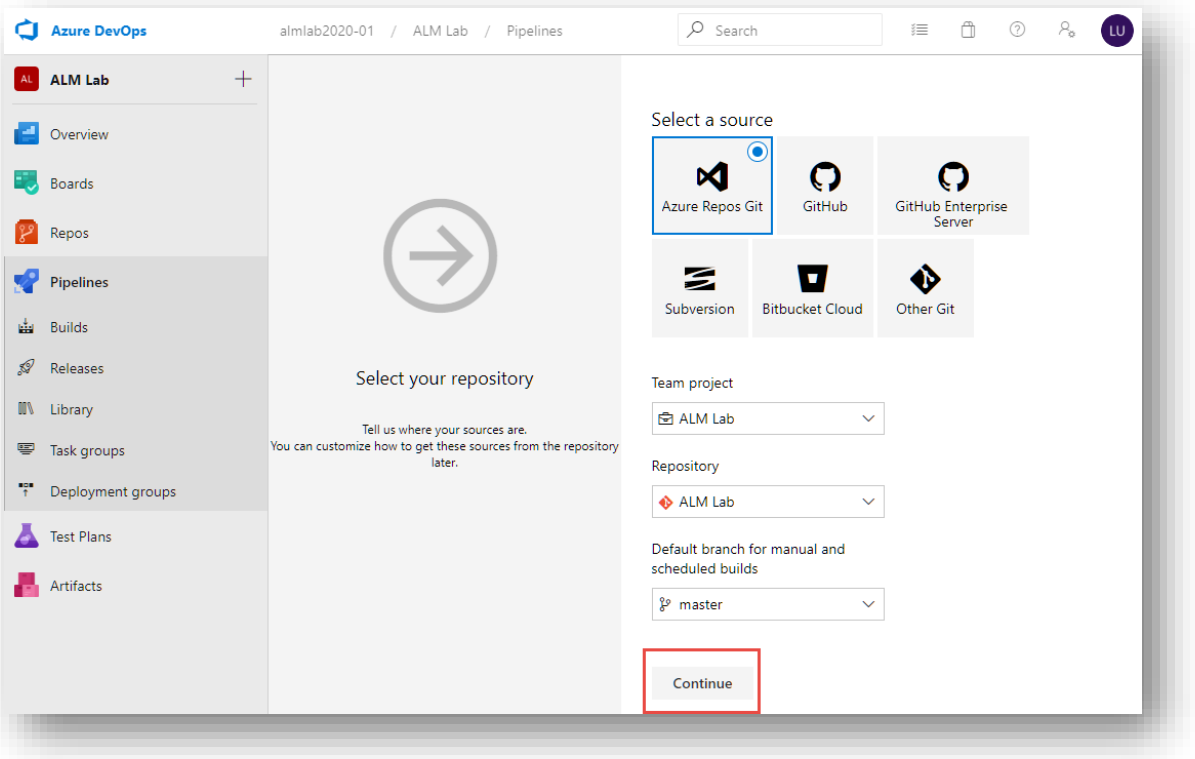
5. In the left-hand navigation, select Pipelines, then click the New Pipeline button to create your first pipeline



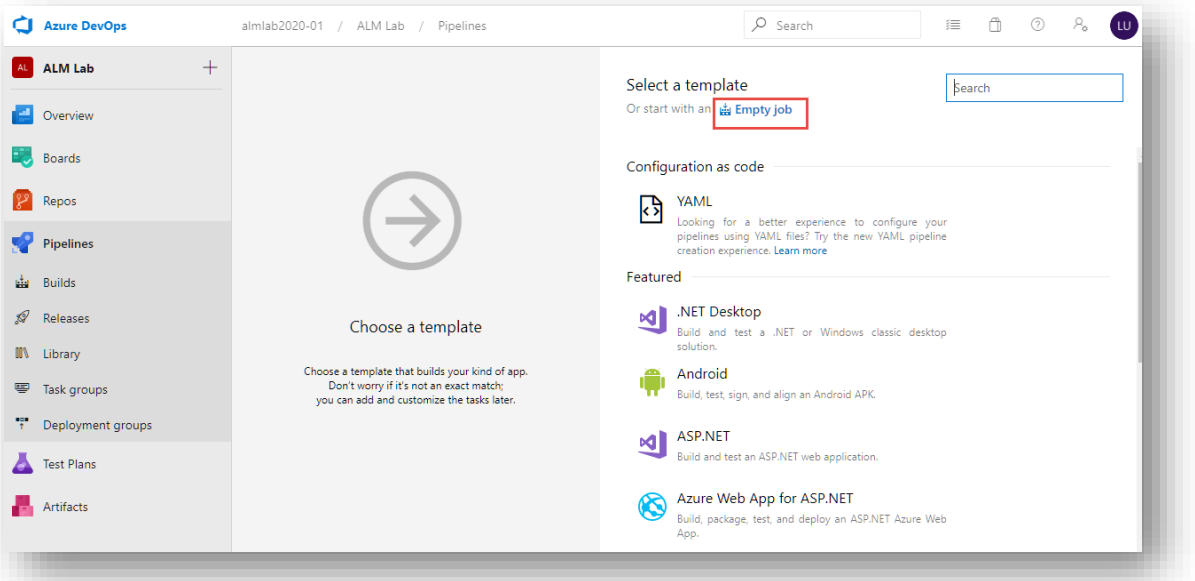
6. Click the Use the classic editor link for your repository



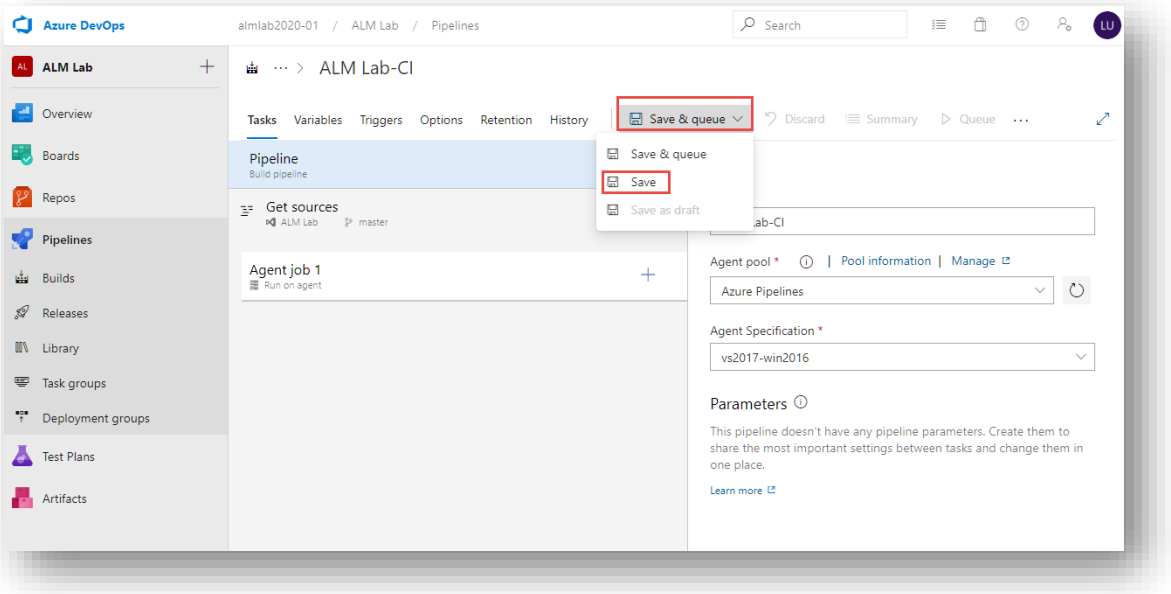
7. Leave the selection on the Azure Repos Git repository and click continue.



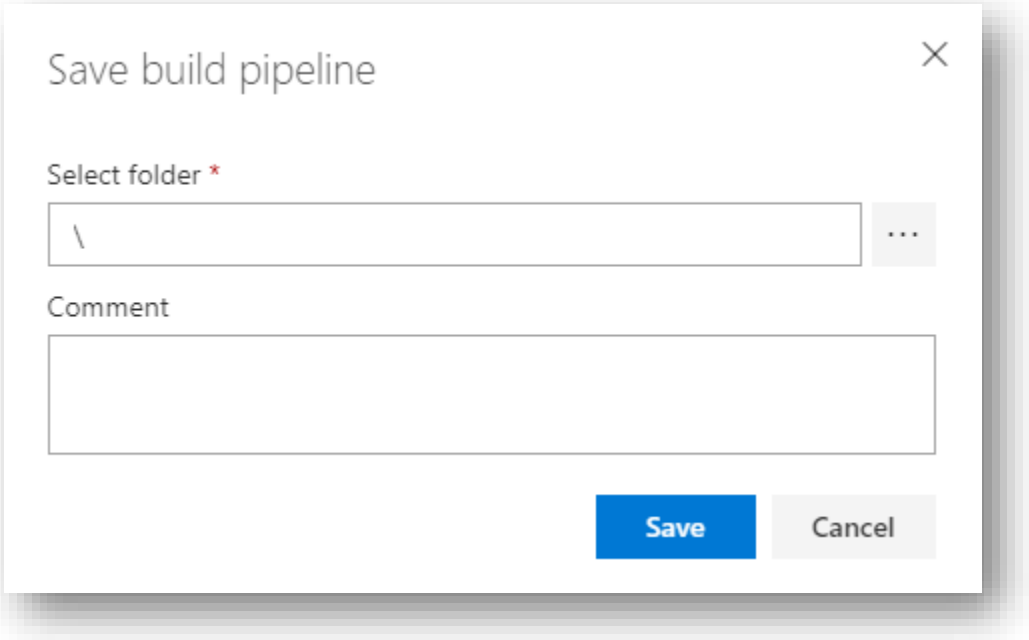
8. Click the Empty Job option



9. Click the Save & Queue menu then click Save



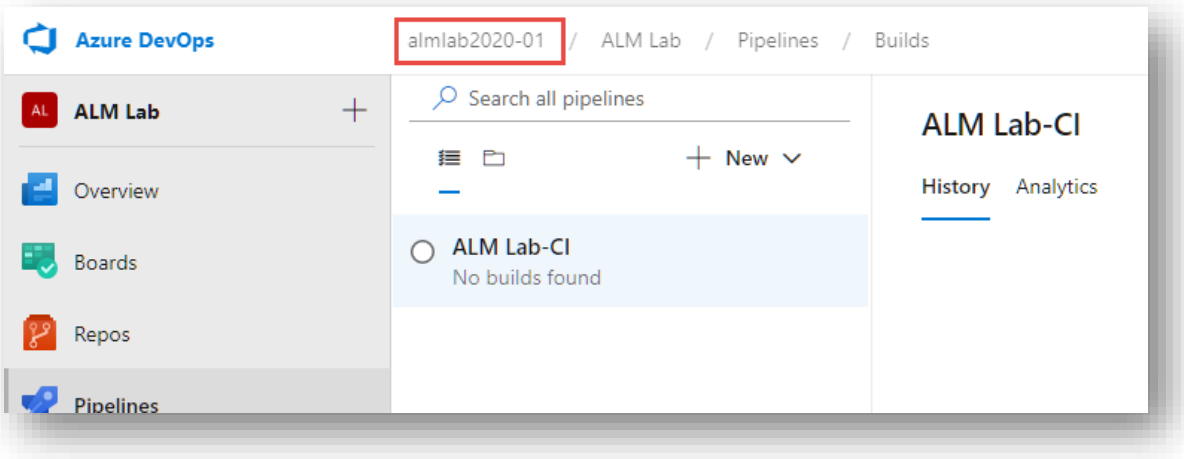
10. Click Save



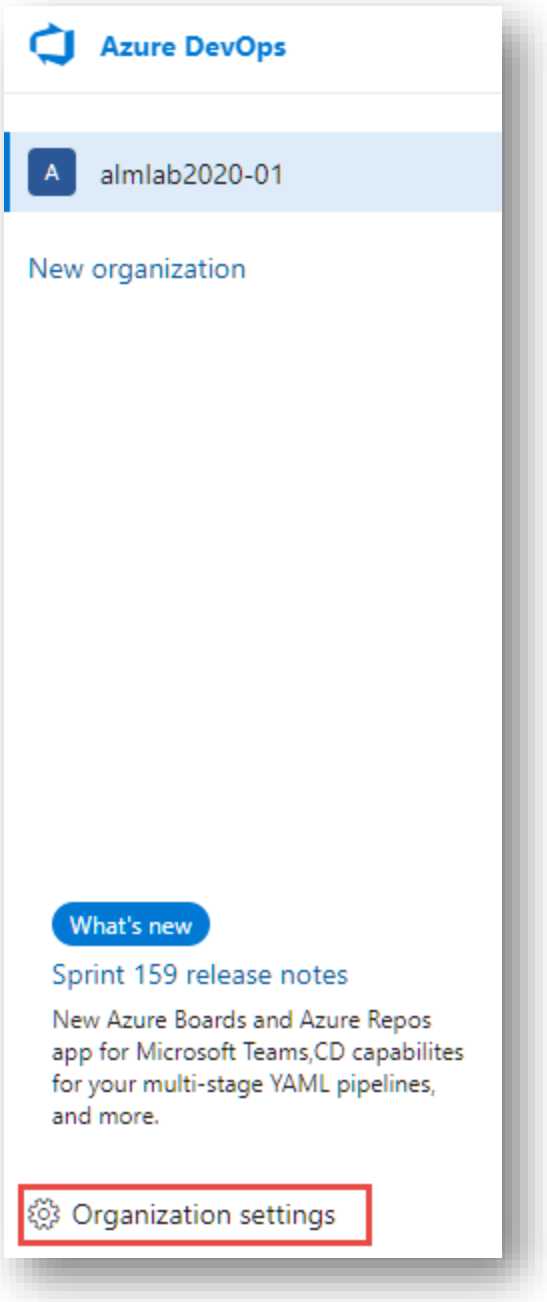
11. This will give you an empty pipeline, which will now enable you to import another pipeline from file. Before we import the pipelines for this lab, we need to install the PowerApps Build Tools.

Install the PowerApps Build Tools

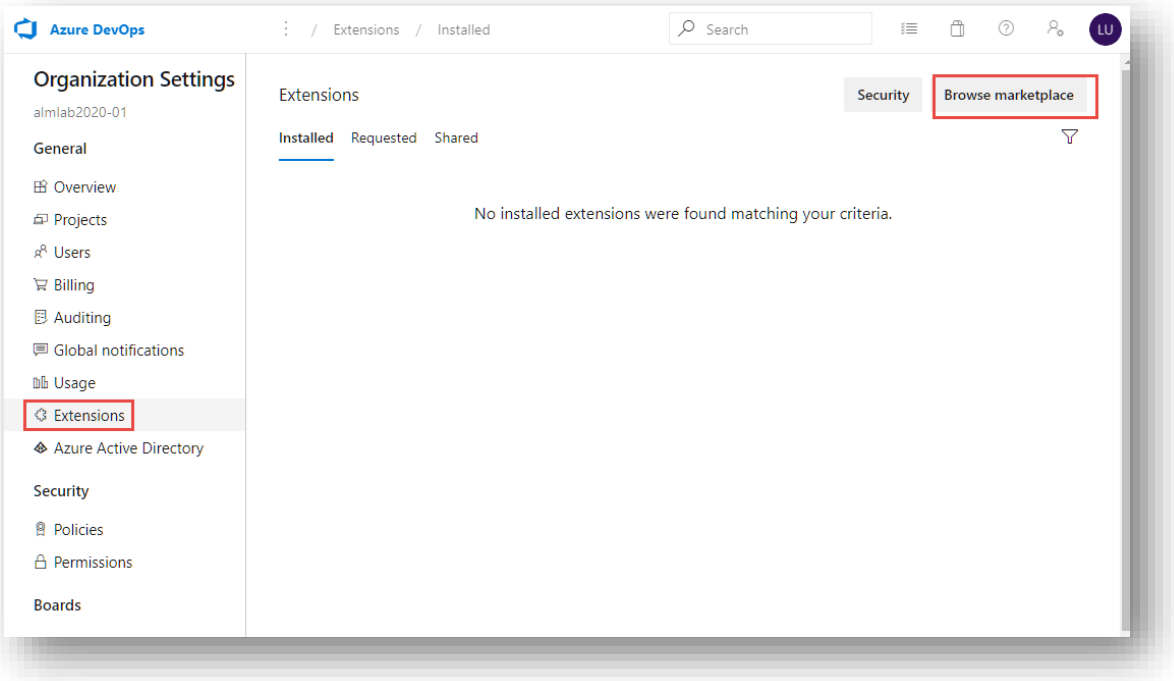
12. Click the organization name as part of the breadcrumbs.



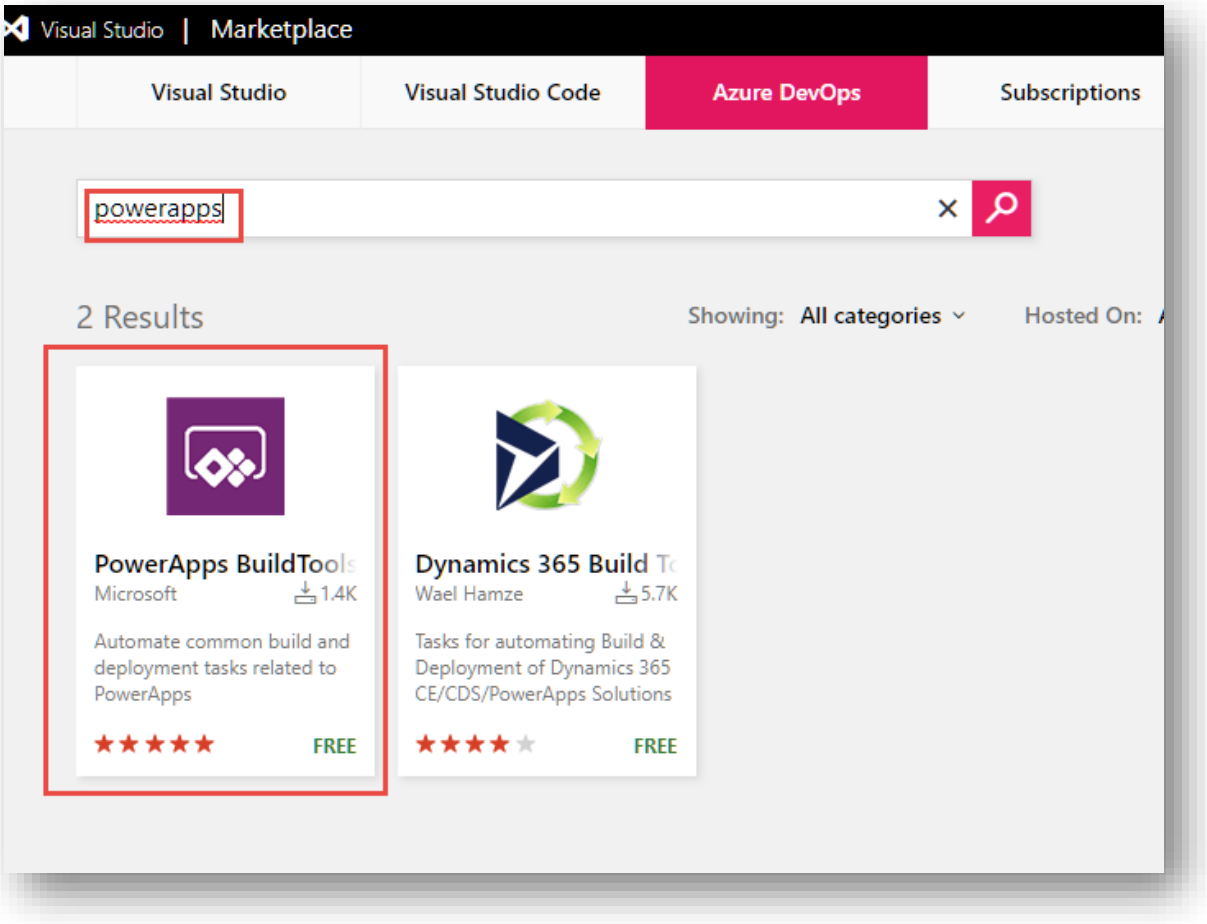
13. Click the Organization Settings button on the bottom left navigation.



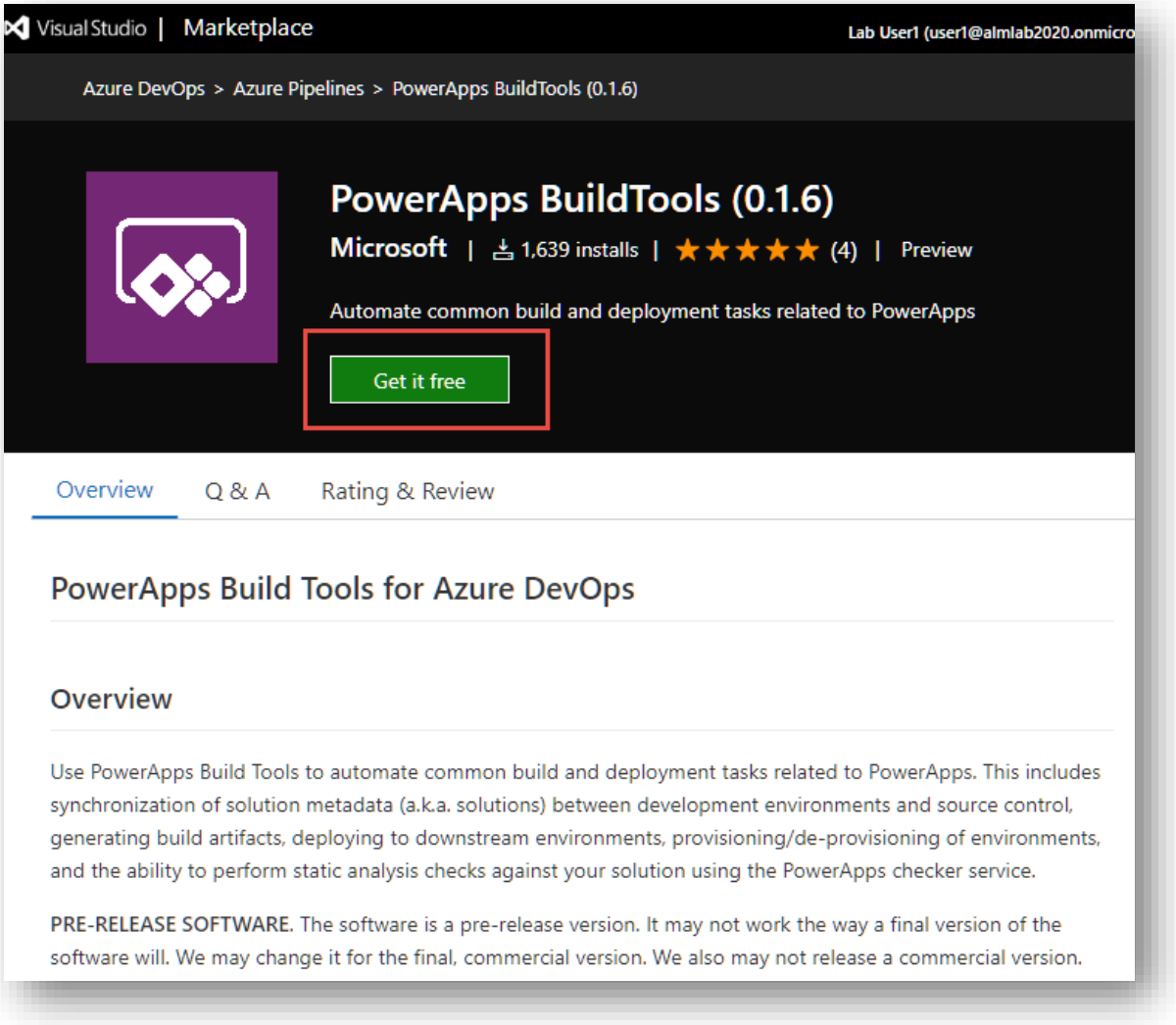
14. On the left navigation, click Extensions, then click Browse Marketplace



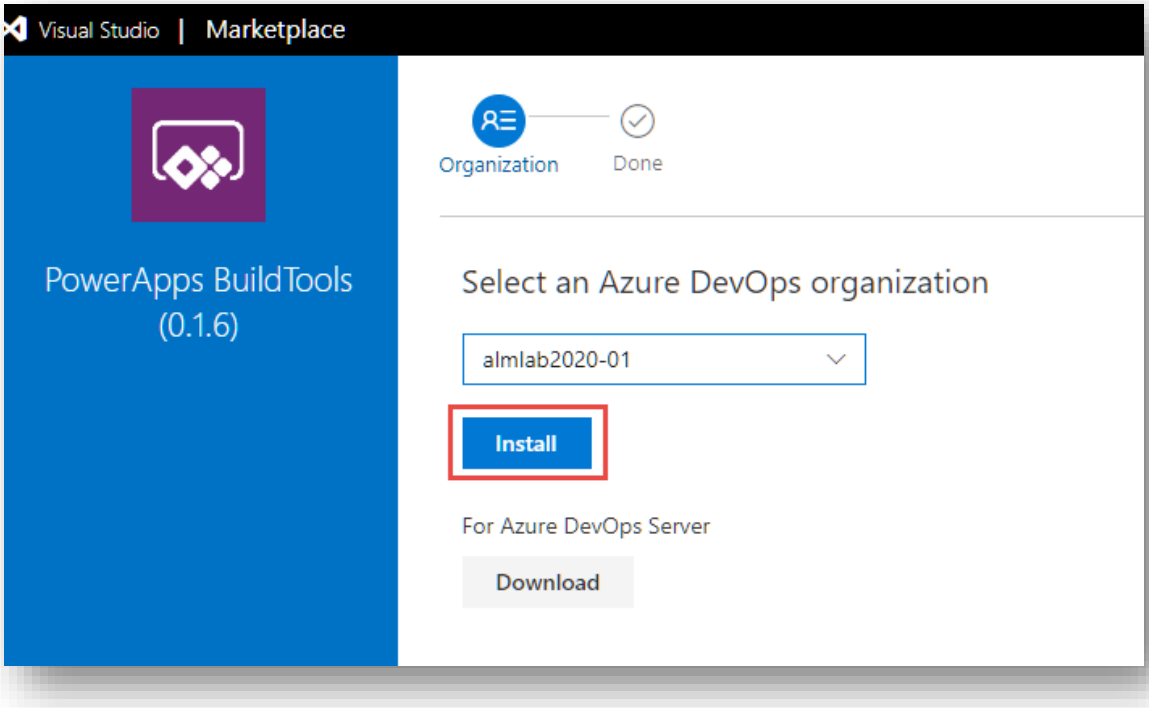
15. In the Azure marketplace, search for PowerApps and select the PowerApps Build Tools from Microsoft



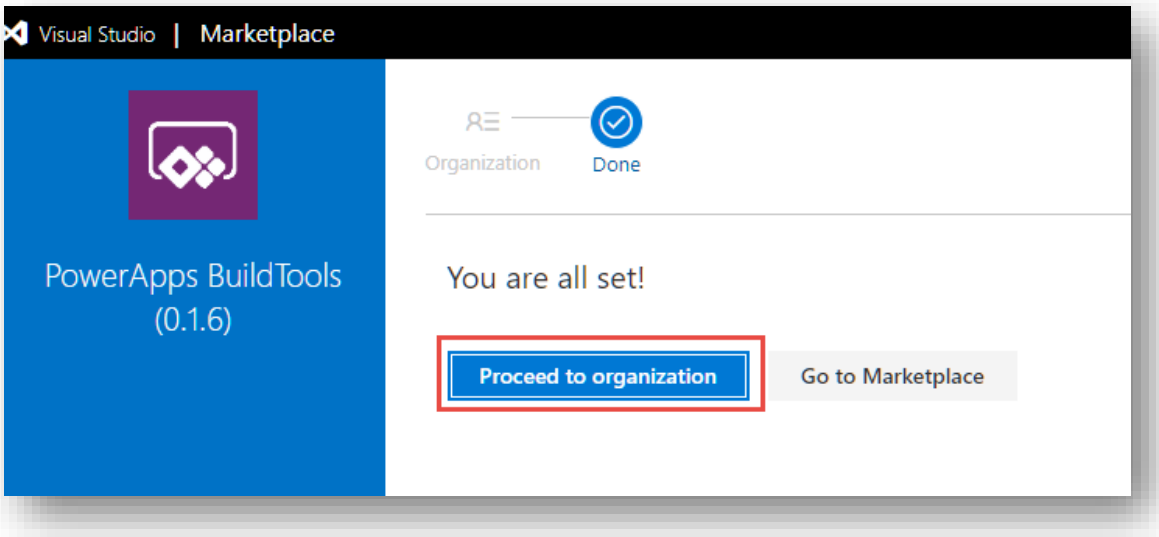
16. Click the Get it Free button



17. Select your org and click install

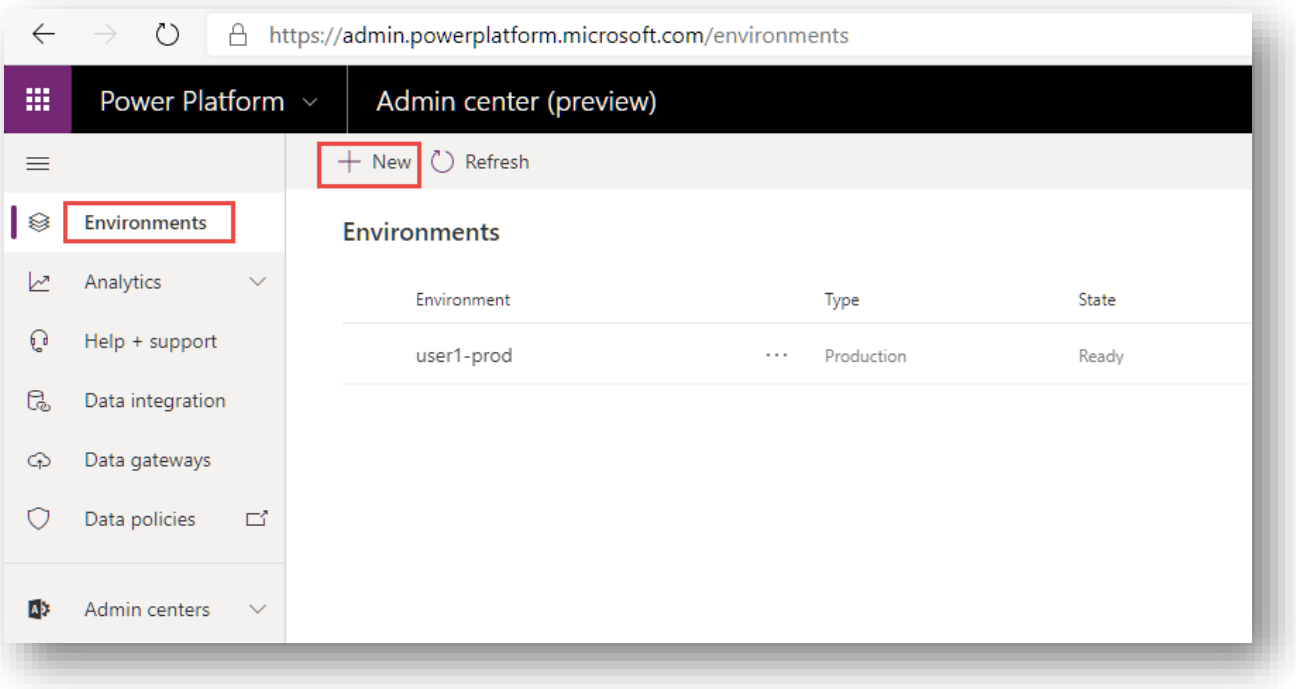


18. Click proceed to organization

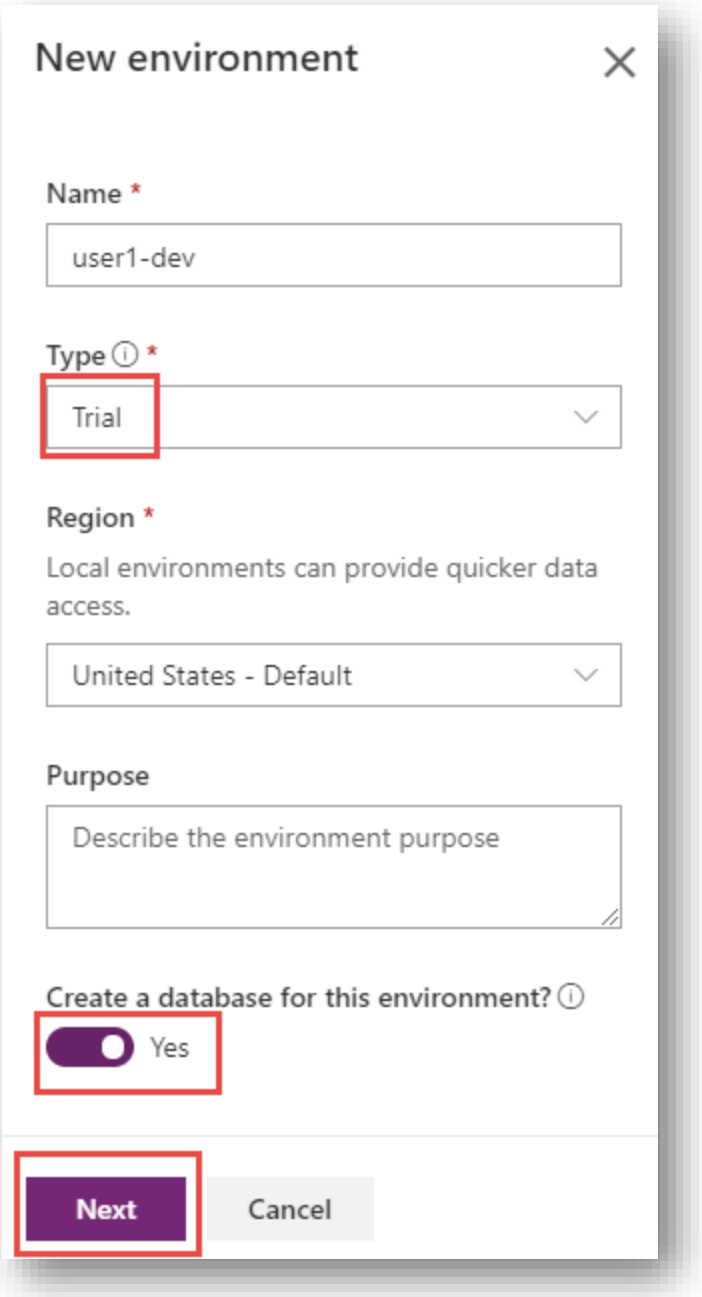


Create Production Environment

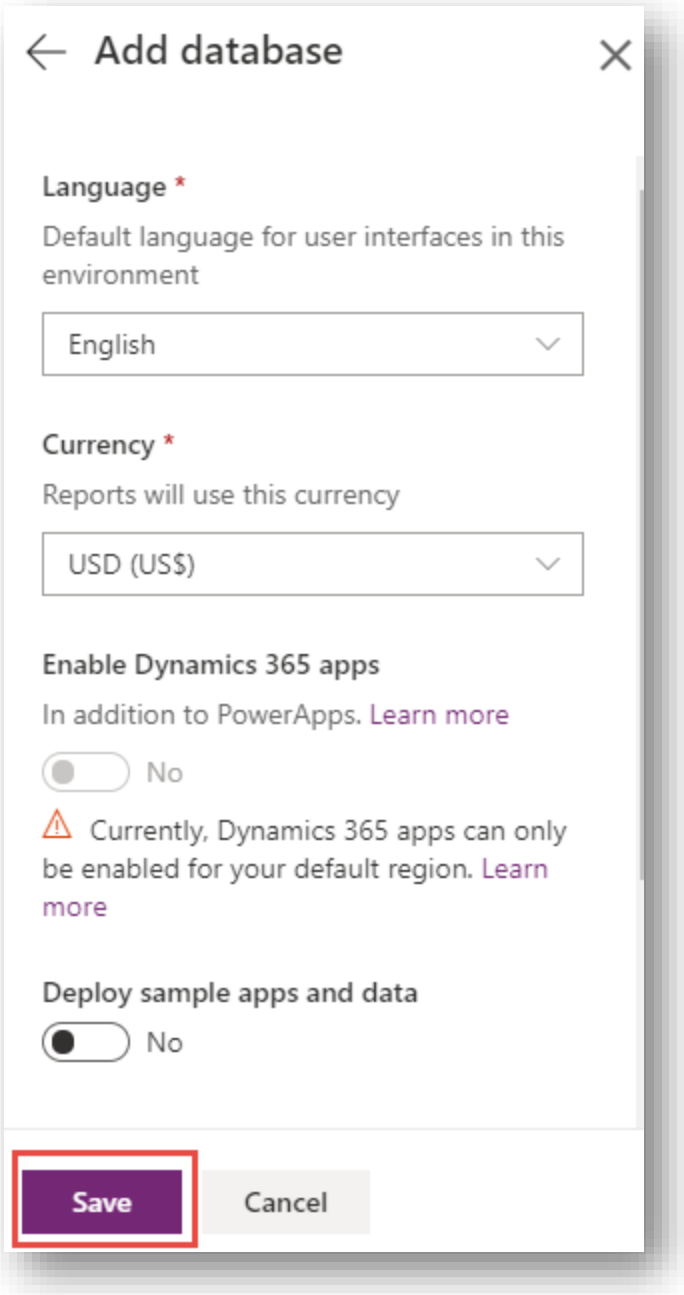
19. Open another tab on your browser and navigate to https://admin.powerplatform.microsoft.com/ then click on the environments tab and the New button.



20. In the panel, give your environment a name – your username- prod (for example user01-prod). Select Trial as the type and turn on the slider to **create a database**.



21. Click **Save** on the next panel to create your environment.

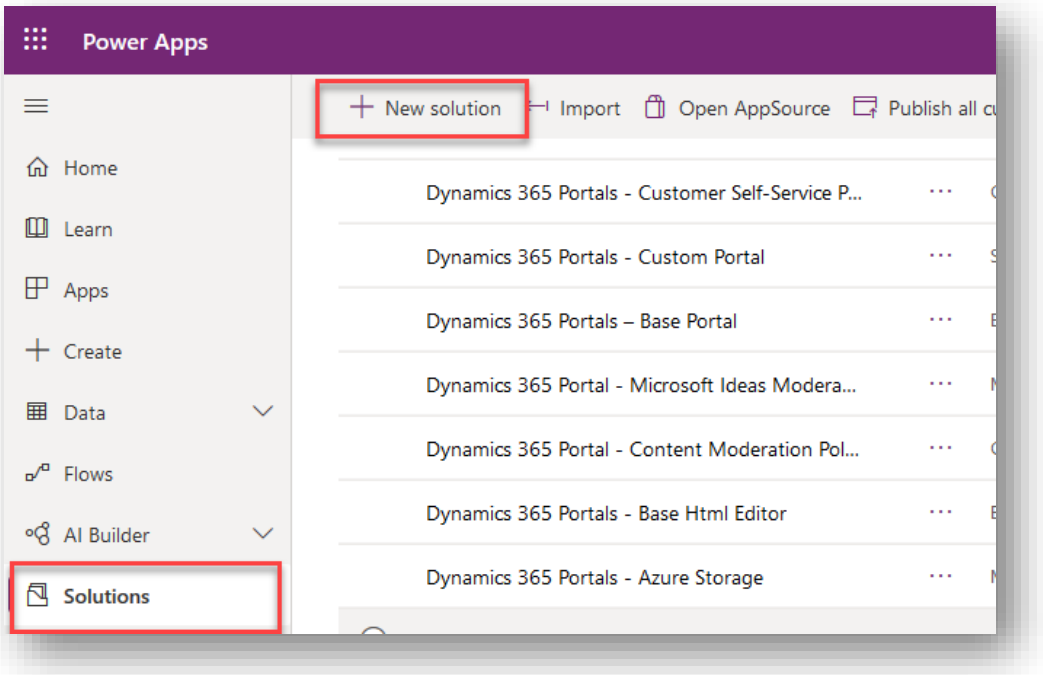


Package your App

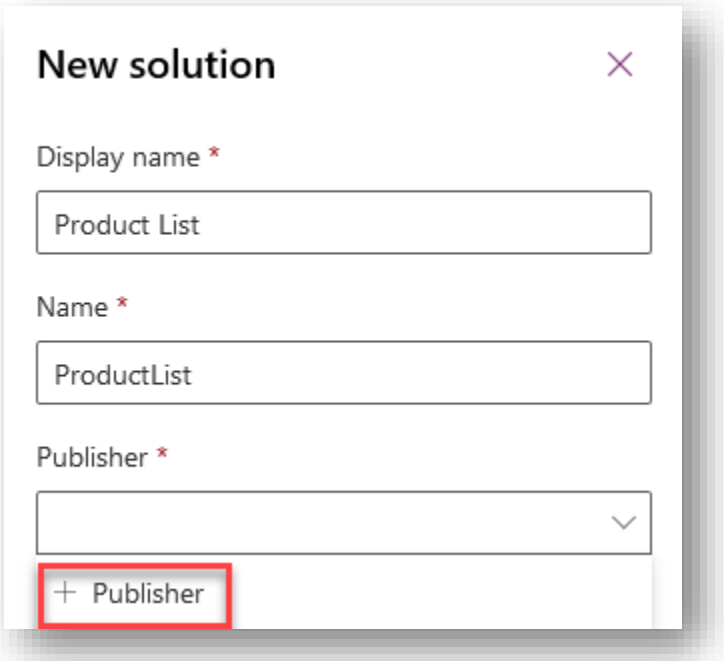
In order to deploy an App from your development environment to downstream environment, first you need to package the App. This is done by creating a new Solution and add the components required by the App to the solution (in this case just the App itself)

Create Solution and Publisher

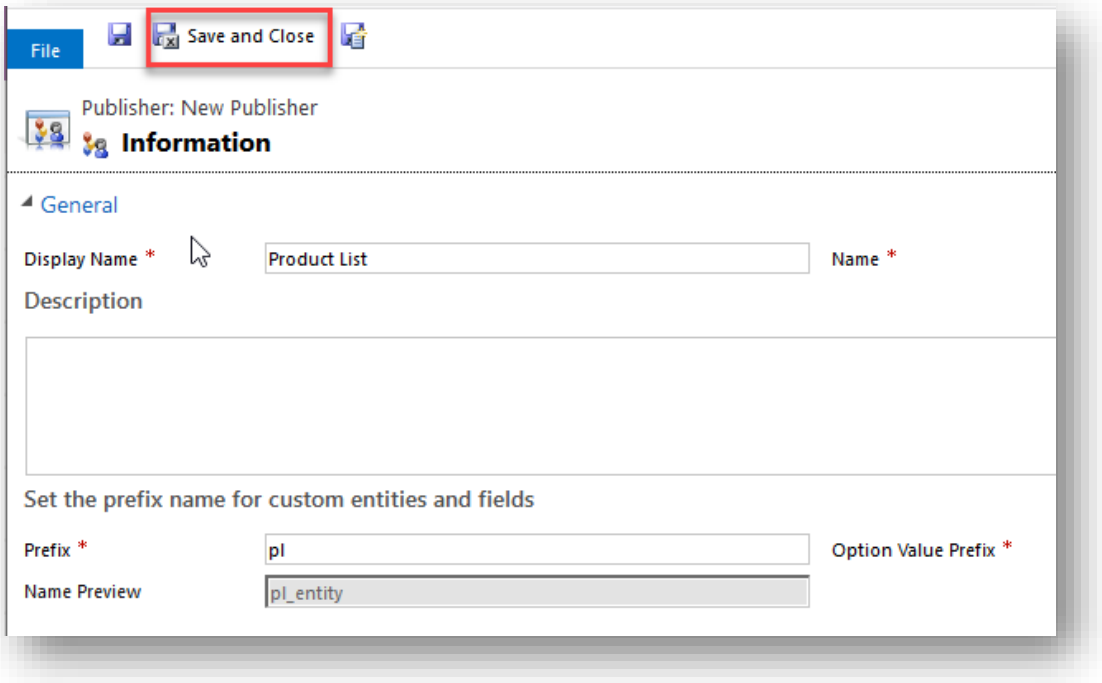
22. Go to https://make.powerapps.com and select **Solutions** and the **New Solution**



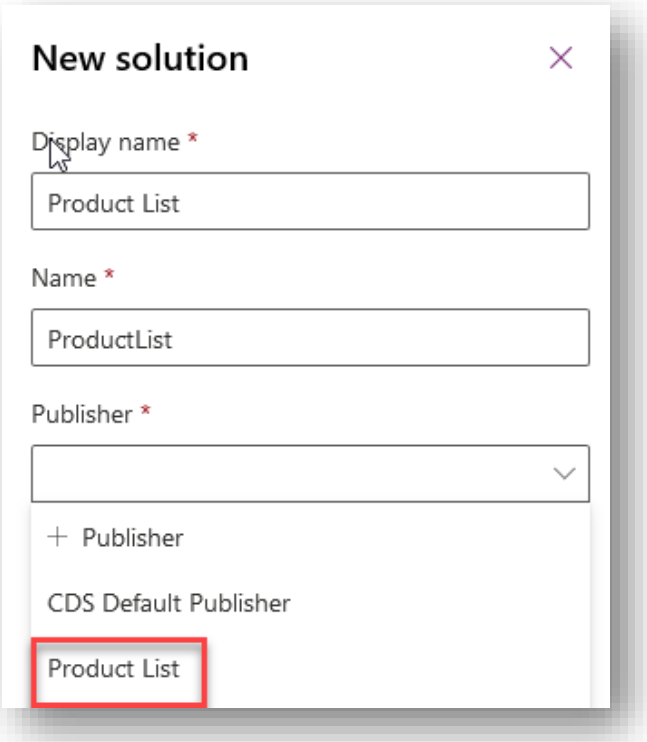
23. In the **New Solution** dialogue, type **Product List** for Display Name (Name will automatically update to ProductList) and then select +Publisher in the **Publisher Dropdown**.



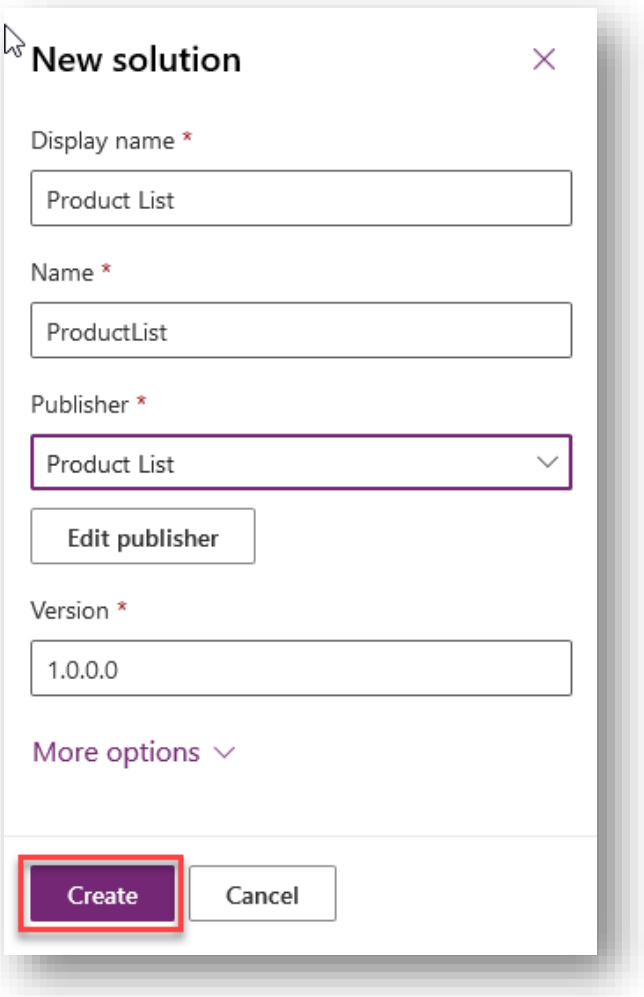
24. Give your publisher a **Display Name**, (auto generated from Display name) and **Prefix**. The publisher and prefix are what uniquely identifies components introduced to the platform
25. Click Save and Close



26. Back in the New Solution Screen, select the newly created publisher (it might take a 20 – 30 seconds for the newly created publisher to be available)

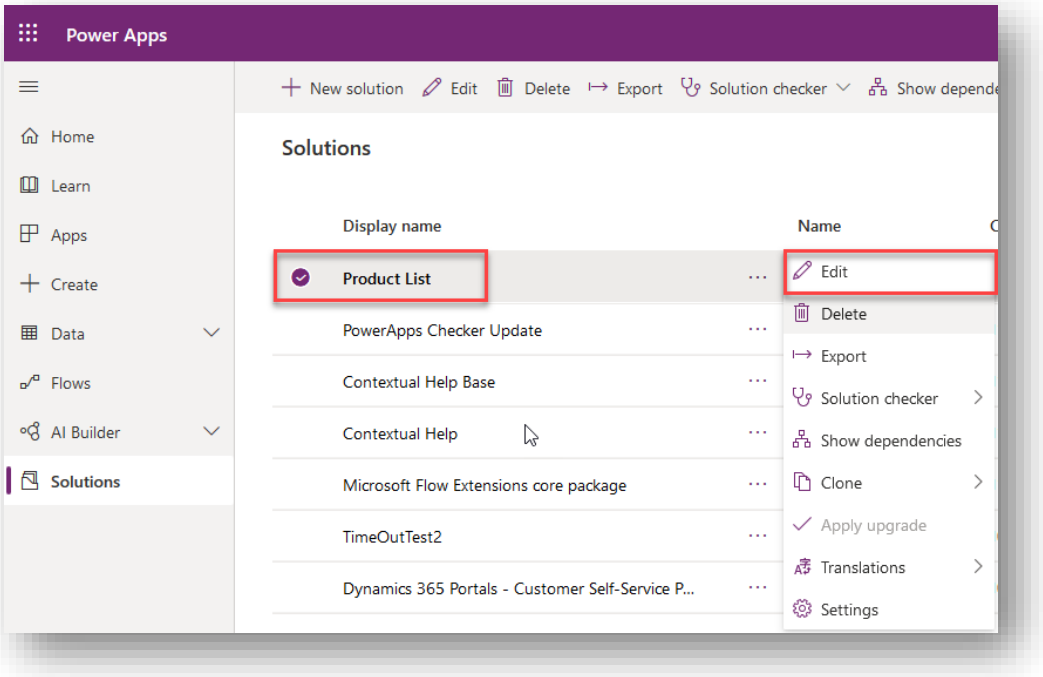


27. Click **Create** to create the solution

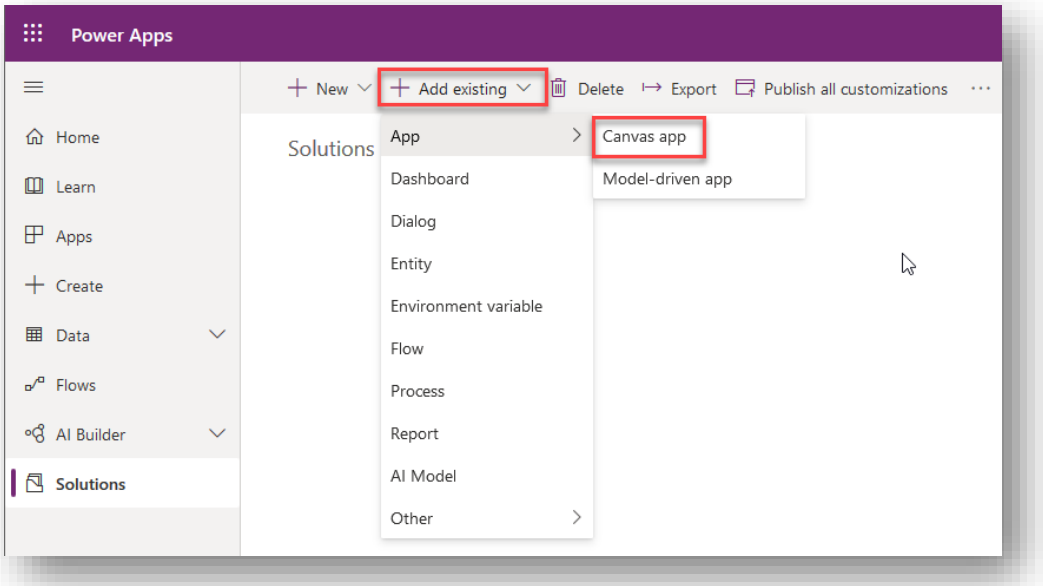


Add App to the Solution

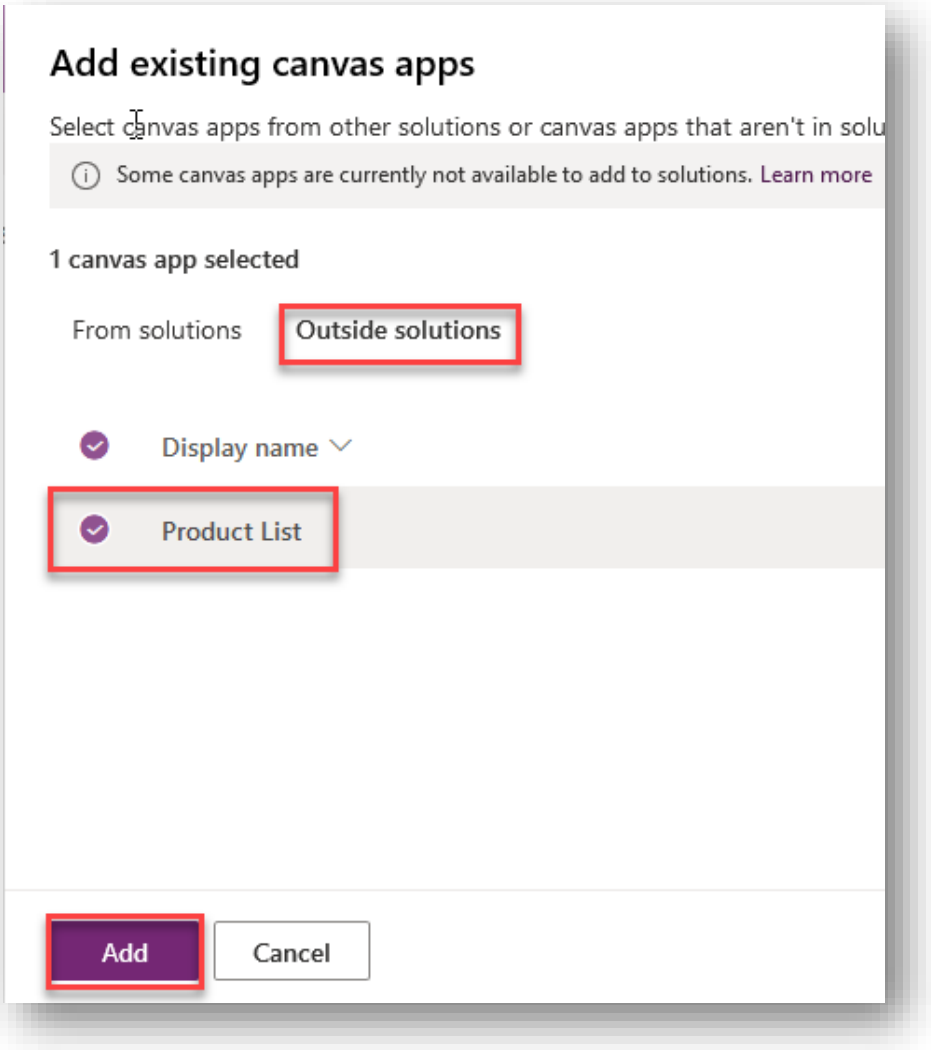
28. Go to https://make.powerapps.com and select **Solutions** and then click **Edit** to open the solution you created in previous step



29. Select **Add existing** and then **Canvas app**



30. On the Add Existing Canvas app Screen, select the **Outside solutions** tab and select the Product List App you built in Section 4 of this lab. Then click **Add**.

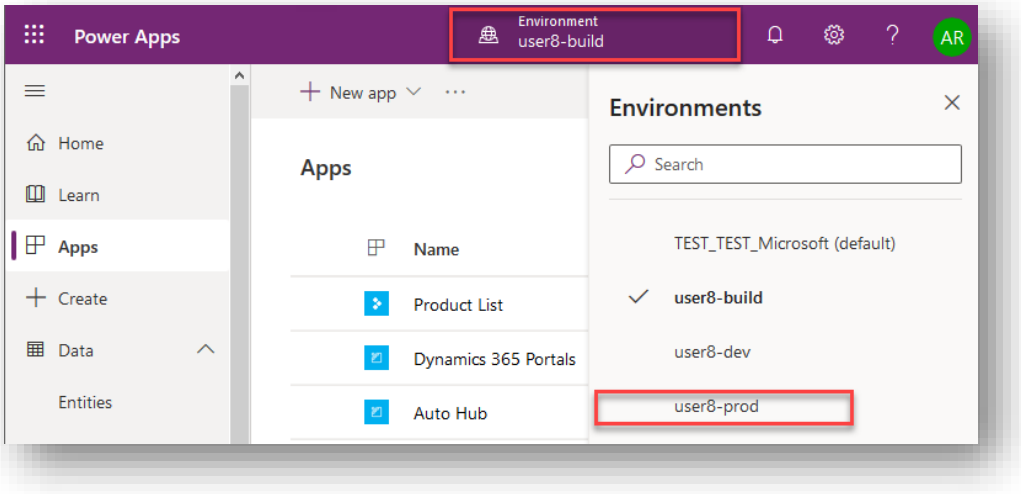


Your app has now been added to the solution and we are ready to use Azure DevOps with PowerApps Build Tools to move the application to source control and setup a pipeline to deploy to downstream environment(s)

Create Custom Connector in downstream environment

NOTE: in future releases this step will not be required as the custom connector can be added to the solution similar to how we added the App in previous steps. This is a temporary manual process but is only required once for each downstream environment.

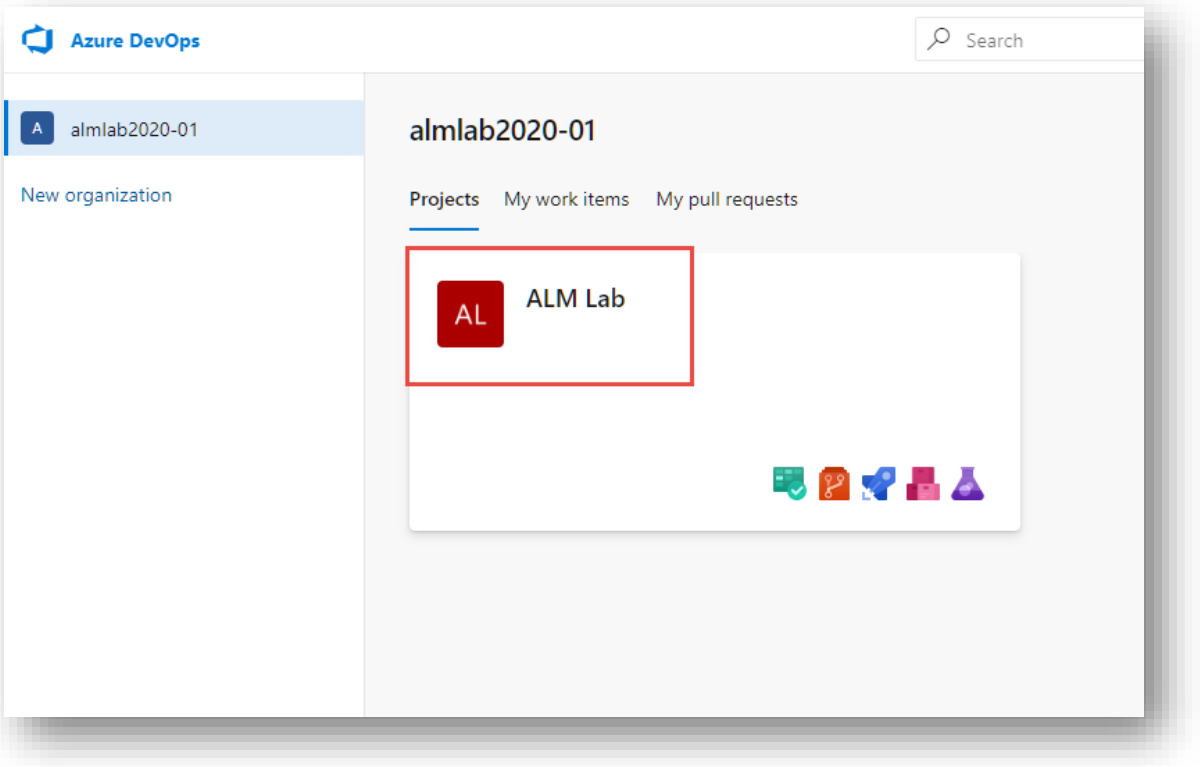
31. Go to make.powerapps.com and select the production environment that you created in Create Production Environment.



32. Repeat exercise 3 to create the custom connector in that environment

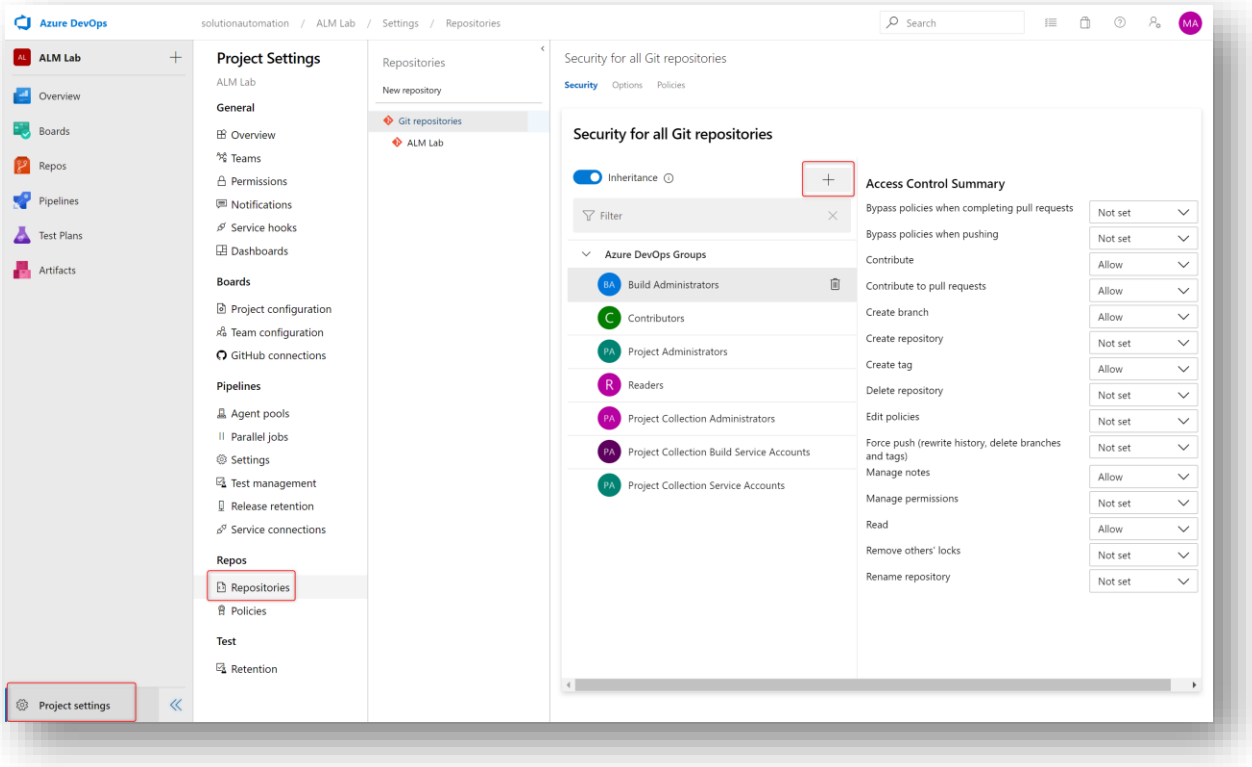
Create Build Pipeline – Export Solution from Development Environment

33. Back in your Azure DevOps subscription (https://dev.azure.com), click your project

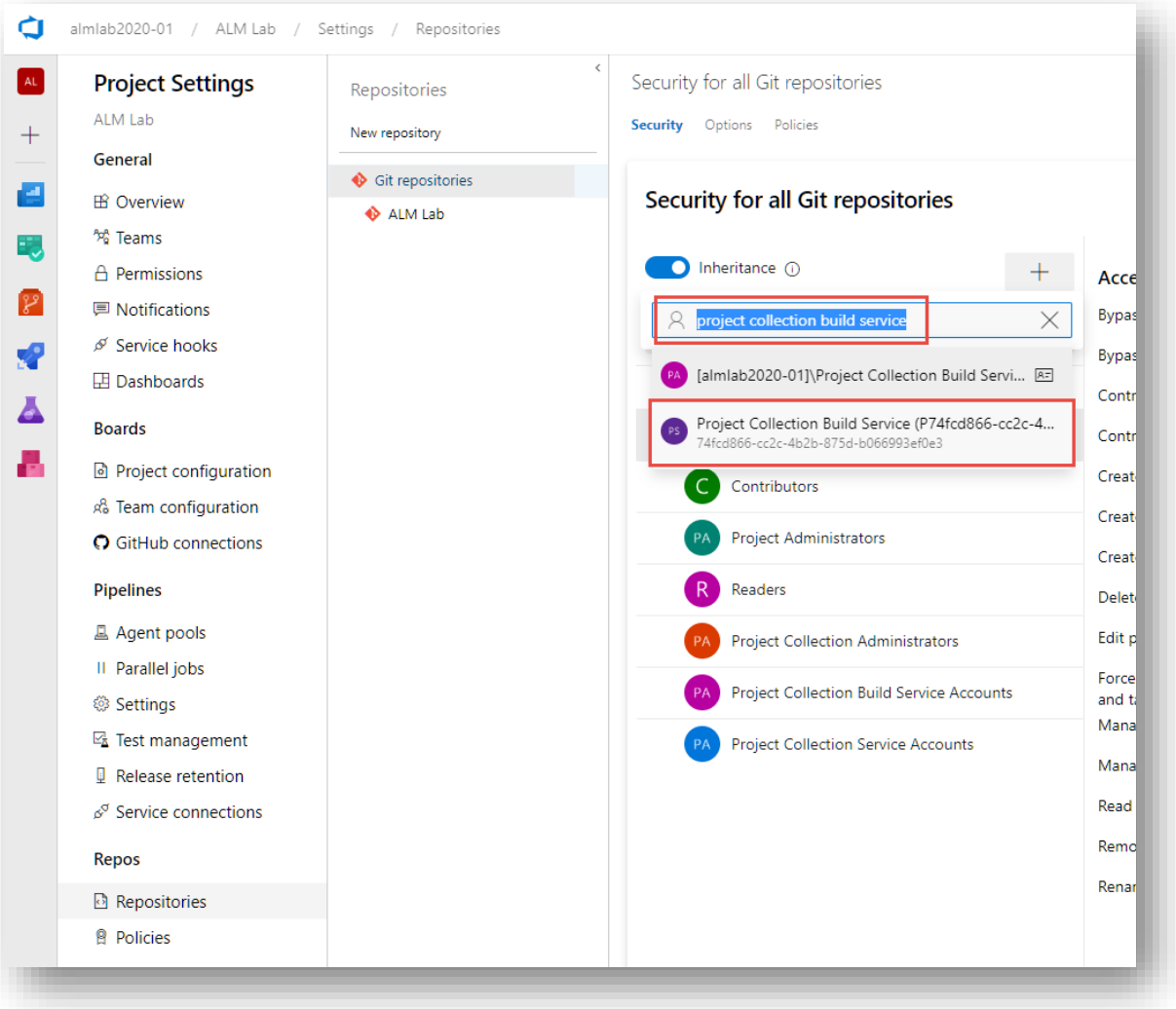


34. The build pipelines that we set up later will be exporting your packaged app from your development environment and check it into your source code repo. This is not a default permission of Azure DevOps, so we need to configure the appropriate permissions for the pipeline to function properly.

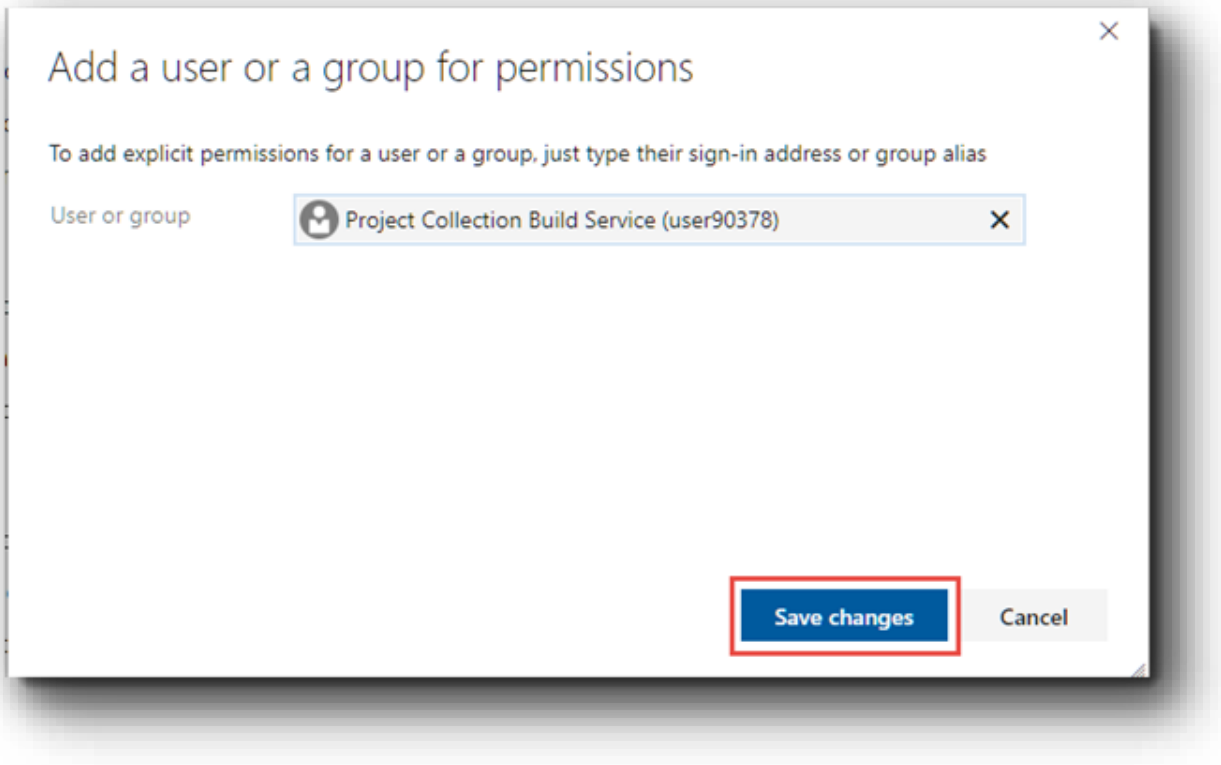
Click the Project Settings icon on the lower left of the screen and click the **Repositories** link in the fly out menu. Click the **Add** button to add a group to define these permissions.



35. **Search** for **Project Collection Build Service** and select it (select the project collection Build not prefixed with the username)

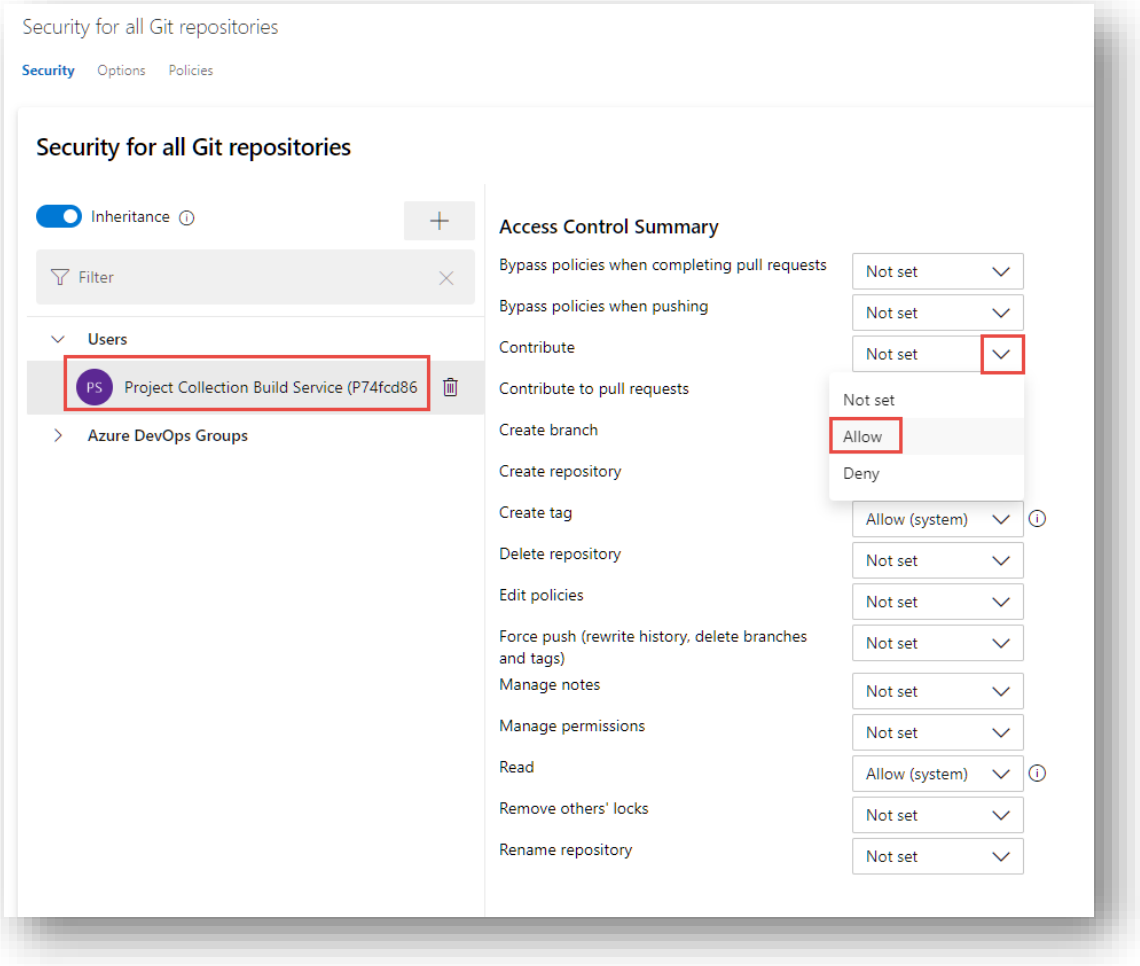


36. Click Save changes

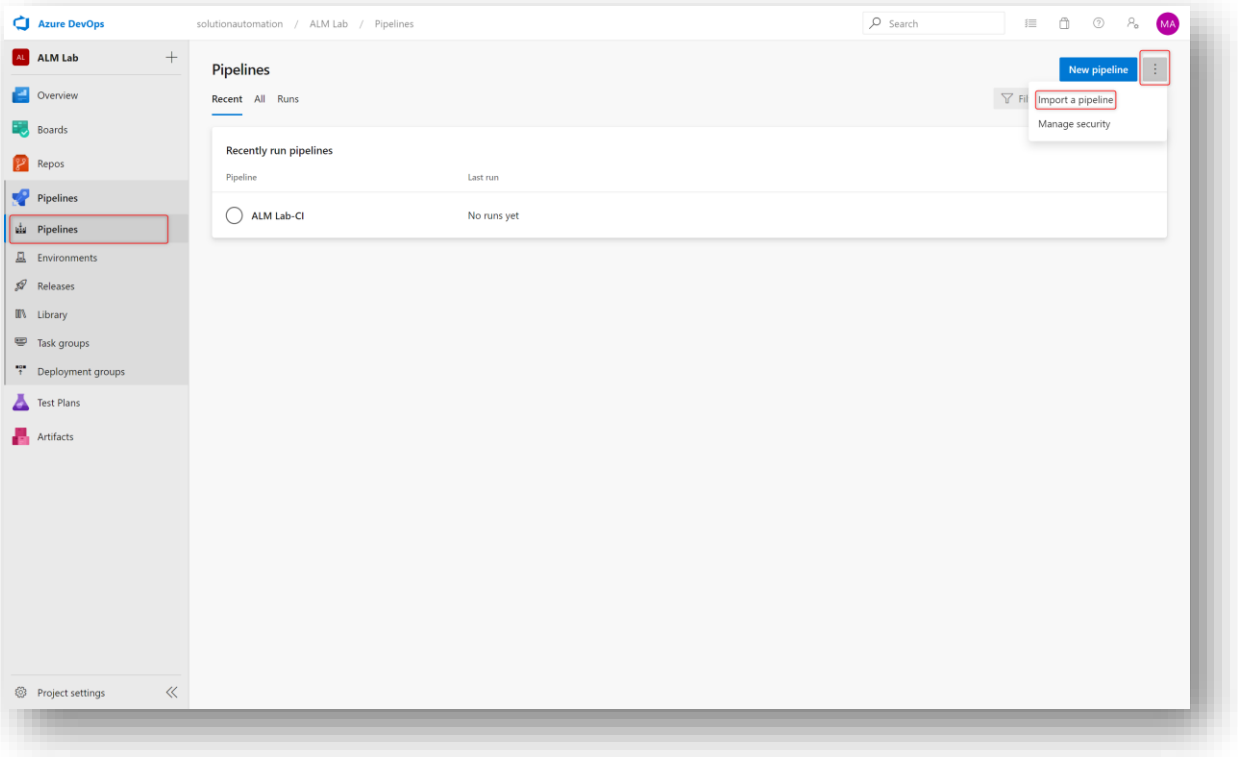


37. Locate the new user you just added and change the Contribute permission from Not set to Allow. Make sure the changes are saved – there will be a green check mark by the drop

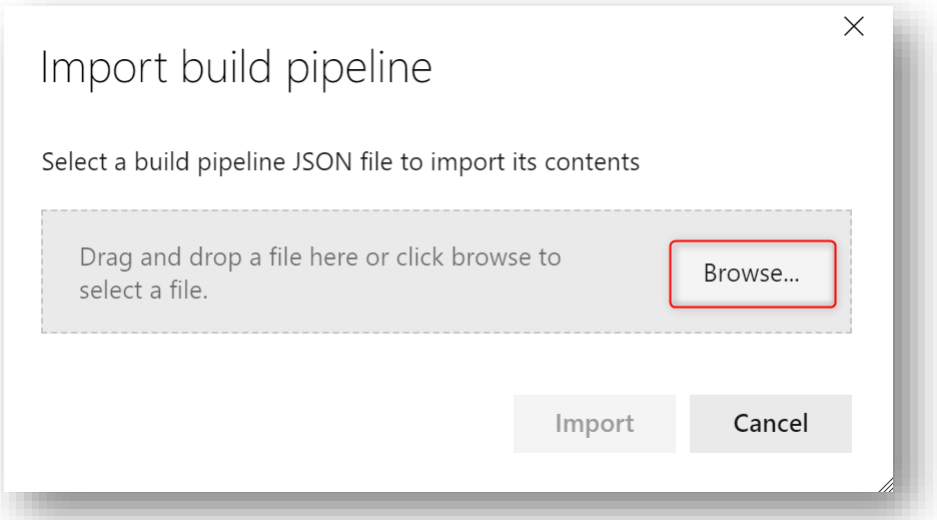
down.



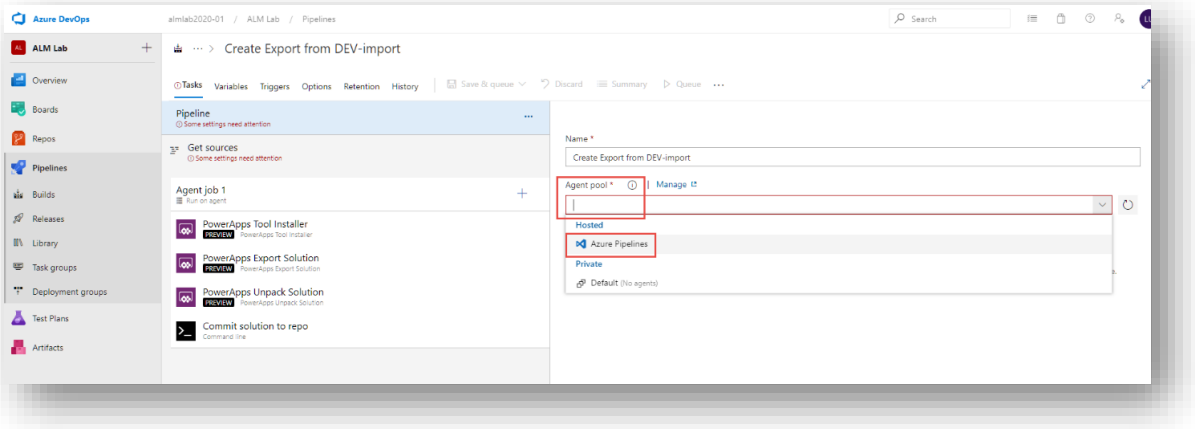
38. Navigate back to Pipelines and select **Import a Pipeline**



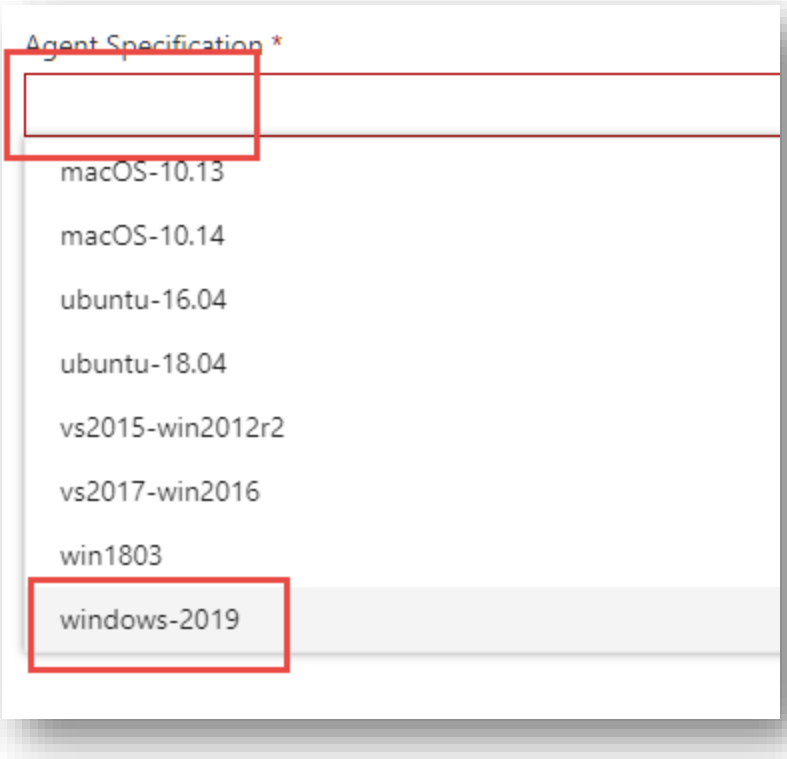
39. Click **browse**, then **select "Export From Dev.json"** from the lab downloads, then click **Import.**



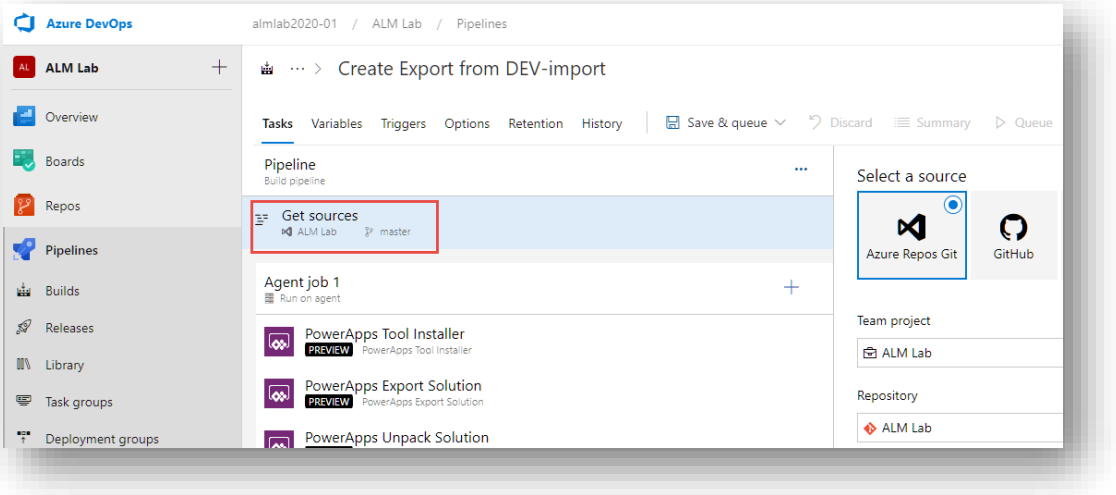
40. The pipeline will import but will need to be hooked up to the azure build agent. Click **Agent Pool** and select **Azure Pipelines** from the options.



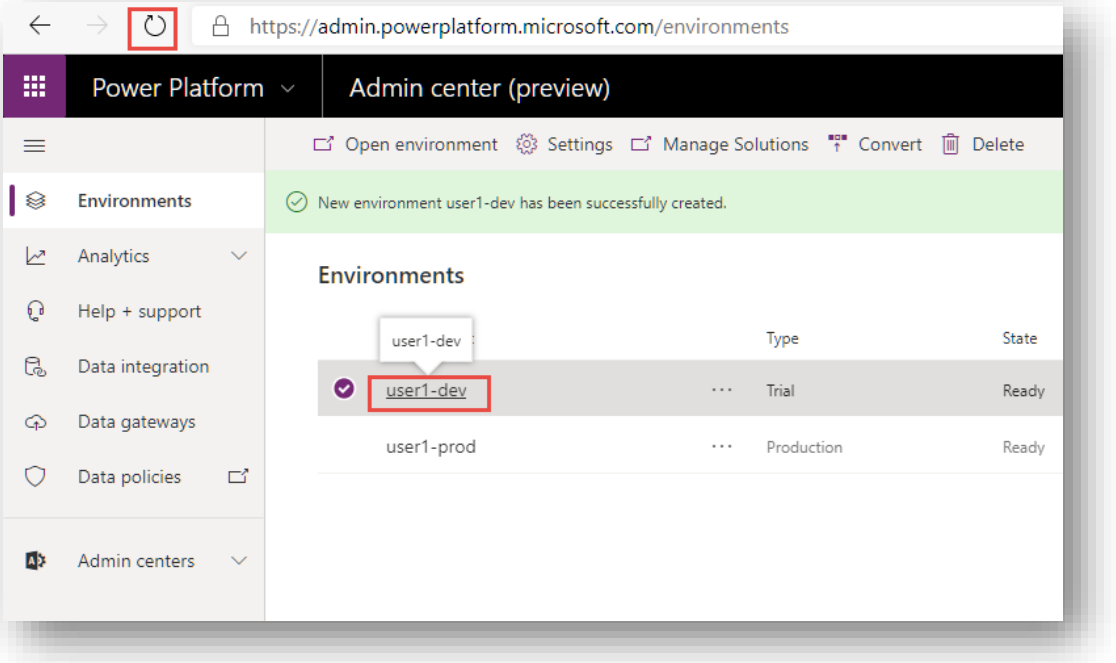
41. Select **windows-2019** for the Agent Specification.



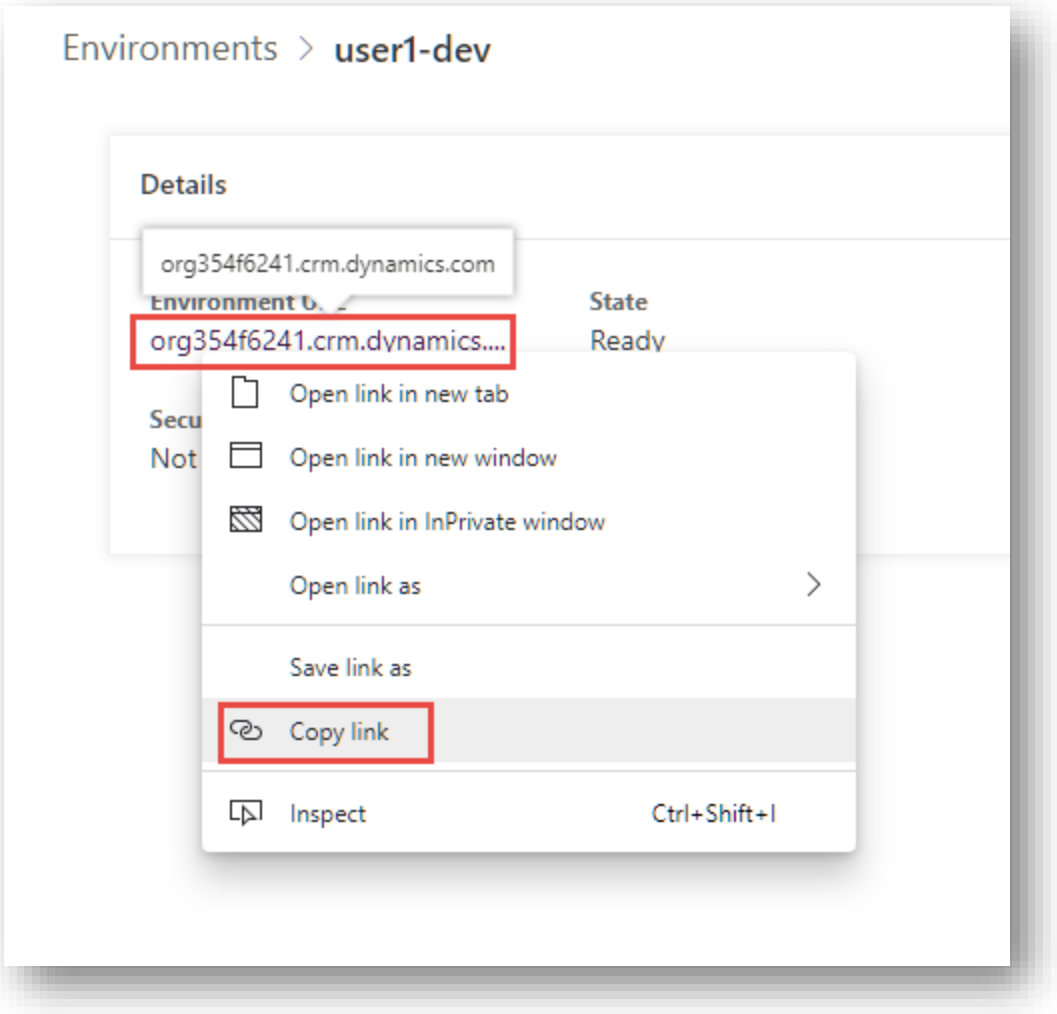
42. Click get Sources to set the missing sources settings (no adjustments need to be made)



43. Now go to the **Power Platform admin center** (https://aka.ms/ppac) and select **Environments** where you will see the a list of available environment. Select the Environment that you used to create the Canvas App and customer Connector in step X.

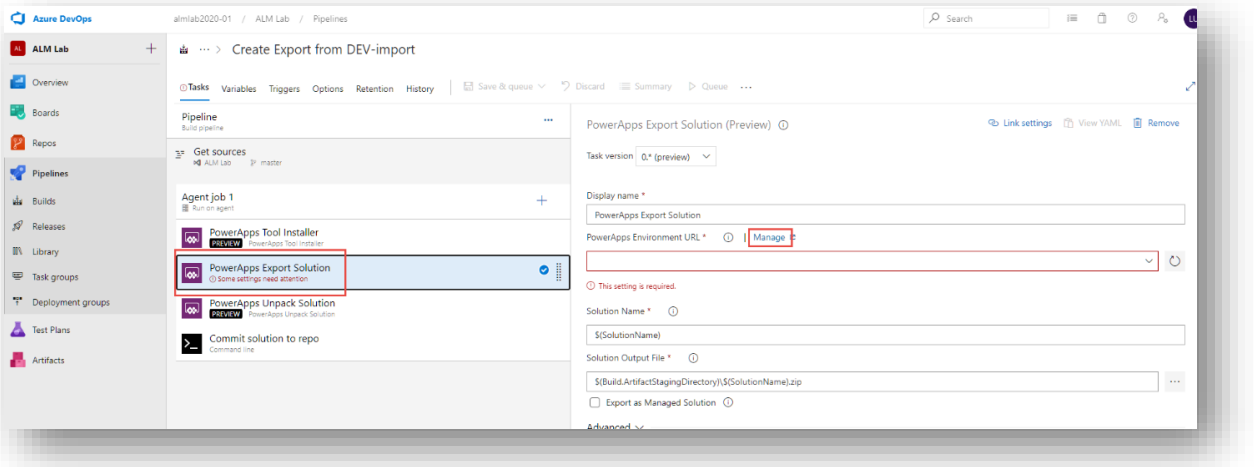


44. Locate the Environment URL, right click and copy it to your paste buffer.

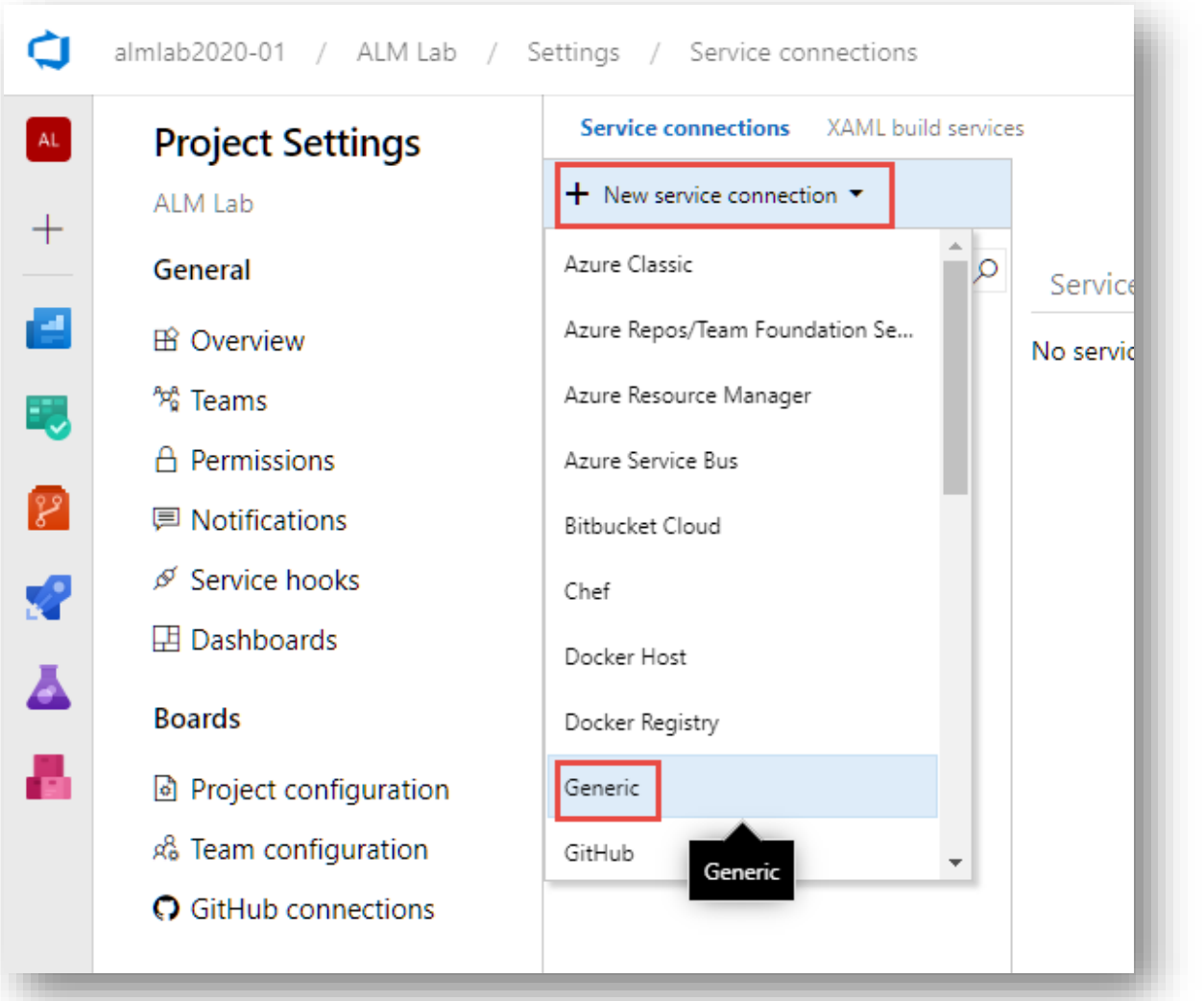


45. Go back to your DevOps tab and select the PowerApps Export Solution task. It will go red because it references an environment connection that does not exist in your environment.

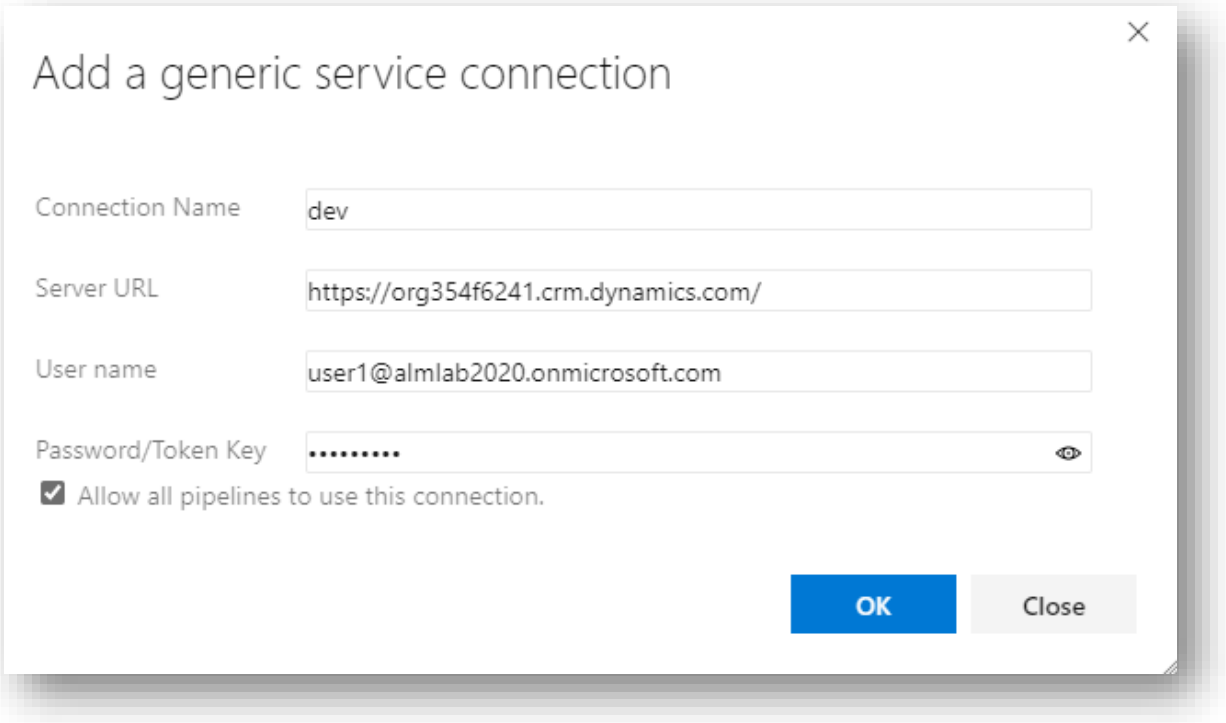
Click Manage to create one.



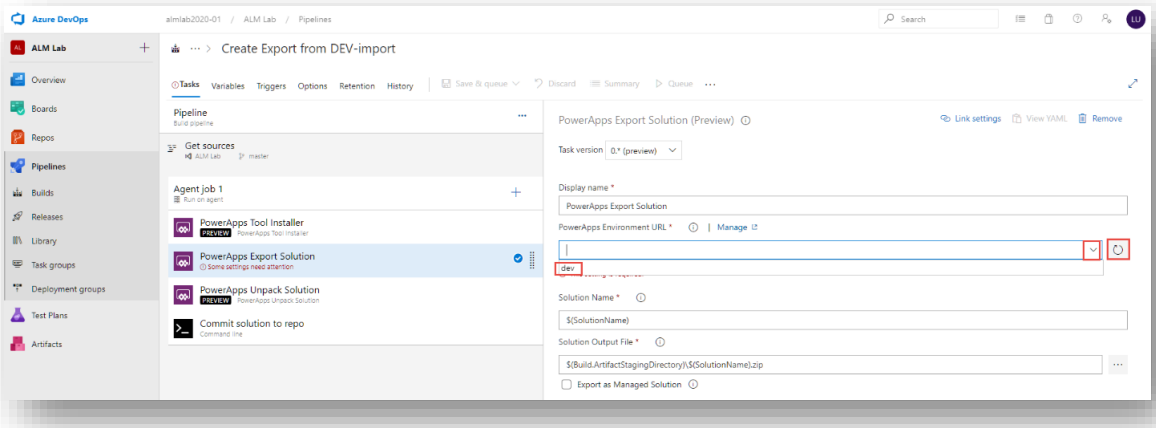
46. Click New Service Connection and select Generic from the dropdown.



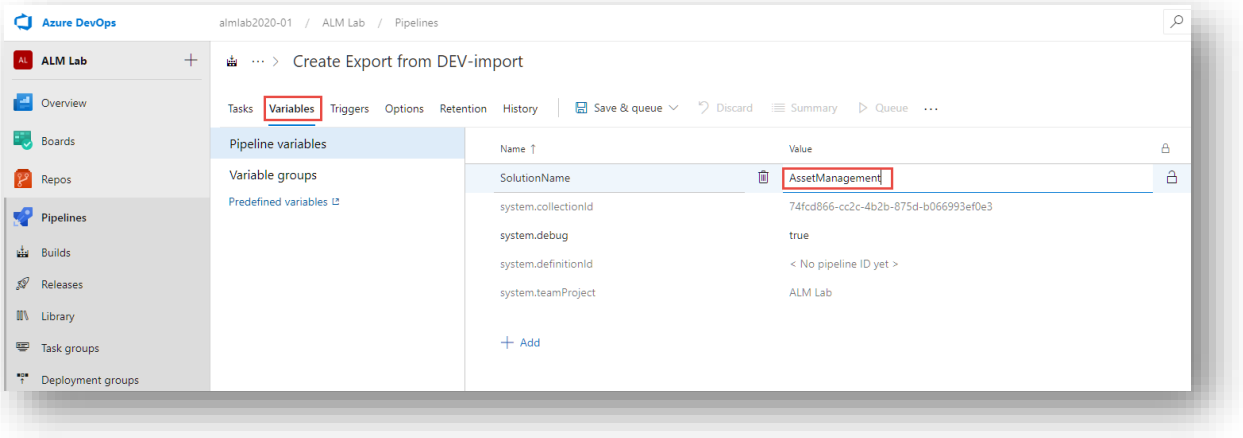
47. Fill in the dialog with dev as your connection name, paste in the server url you copied to your clipboard for your development environment, and add the credentials you are given for the lab and click OK.



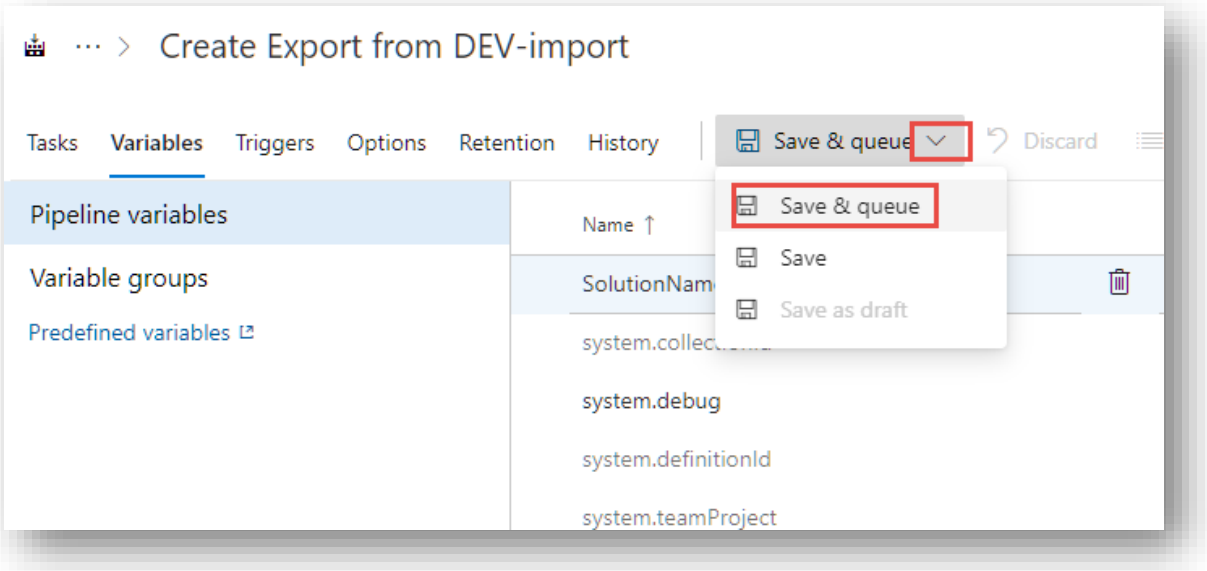
48. Switch back to your pipeline browser tab, click the refresh button highlighted below (NOT F5)- don't refresh the browser!)and select your new connection.

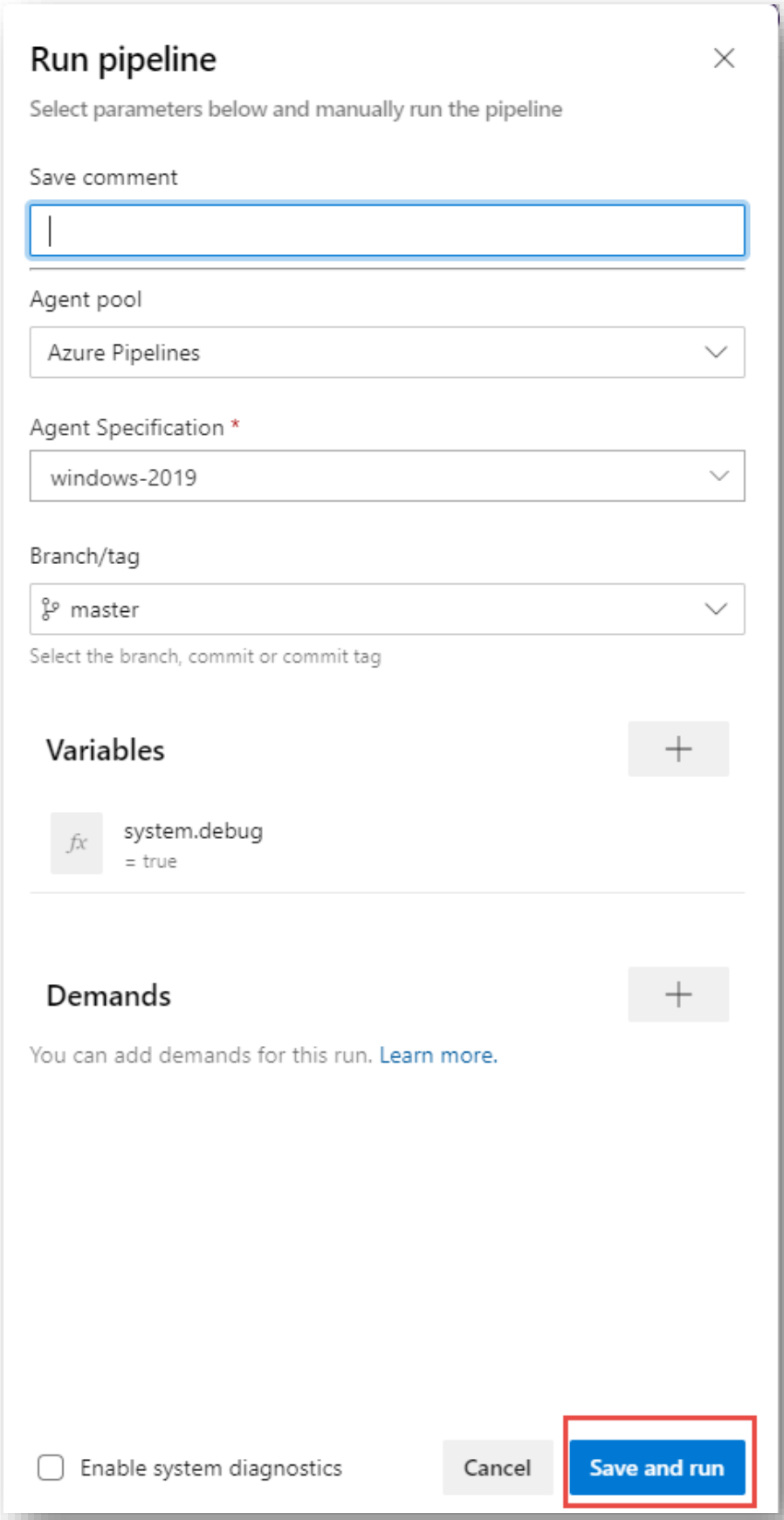


49. Switch back to your pipeline and click the Variables link, then change the SolutionName variable value to ProductList

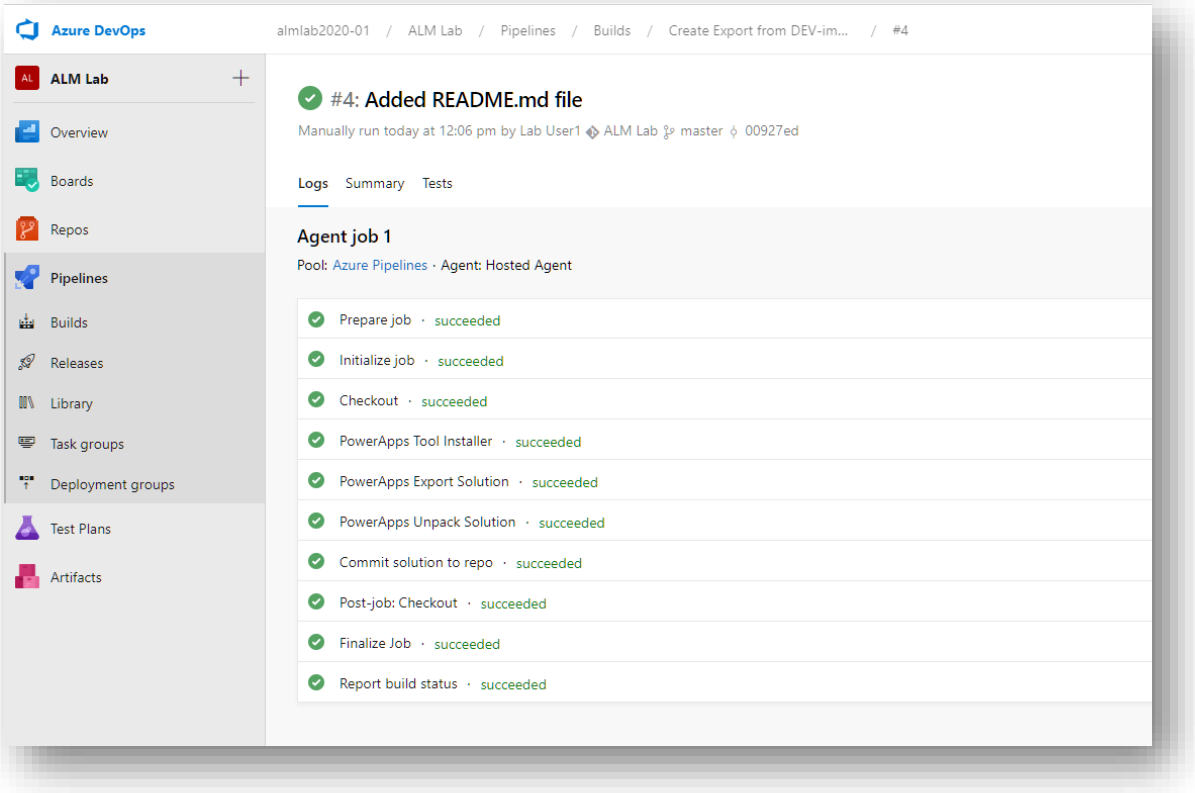


50. Save and Queue your pipeline

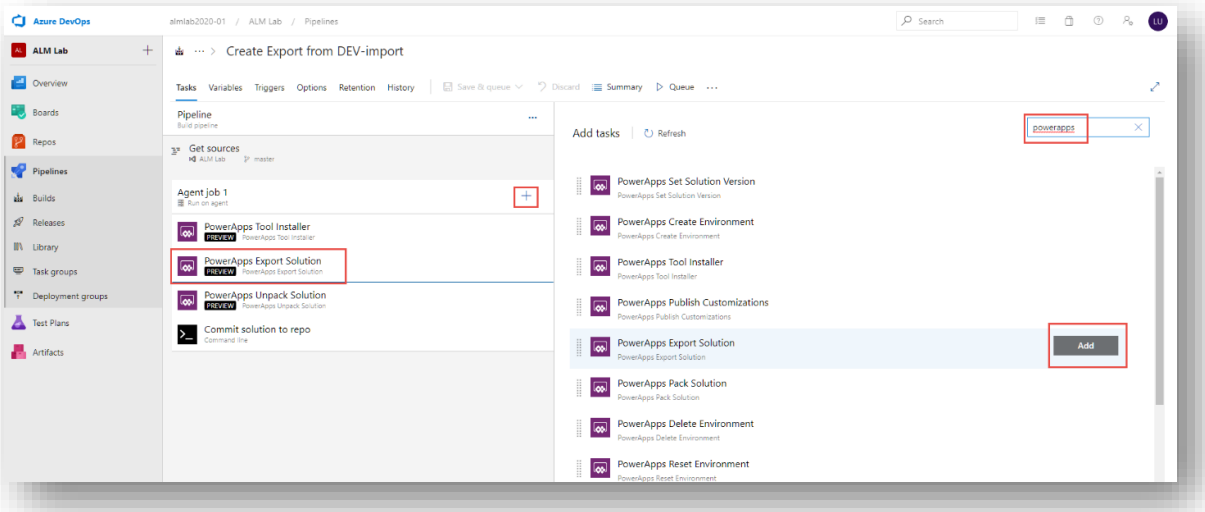




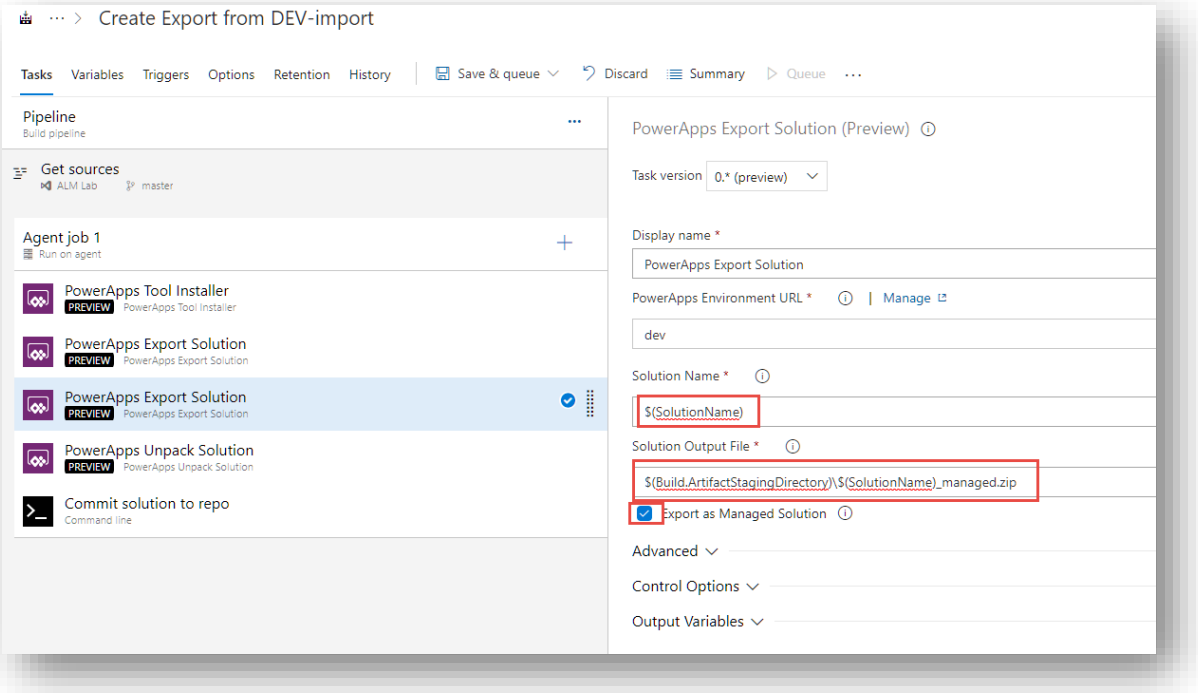
51. Inspect your run log to ensure that everything ran properly. You should see green checkmarks on all steps



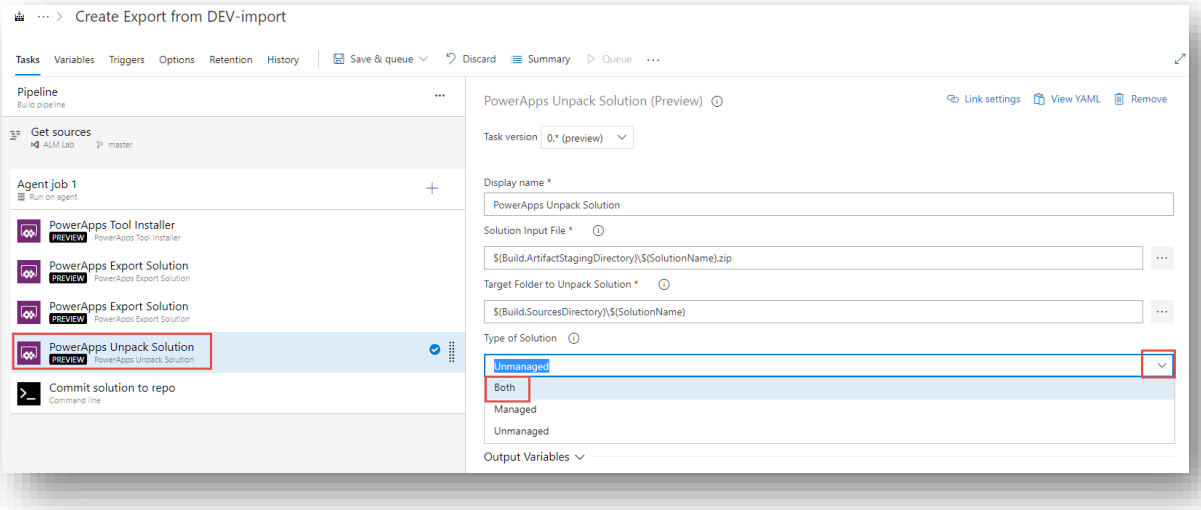
52. Since this lab only has 2 environments and does not have a separate build environment, we are going to modify the pipeline to export both unmanaged and managed files and check them both into source control. Edit your pipeline, select the existing Export task and click the + button to add a new task. Enter PowerApps in the search box and add the PowerApps Export Solution task.



53. For the task properties, use $(SolutionName) for the name and $(Build.ArtifactStagingDirectory)\$(SolutionName)_managed.zip for the filename. Make sure to check the Export as Managed Solution option.



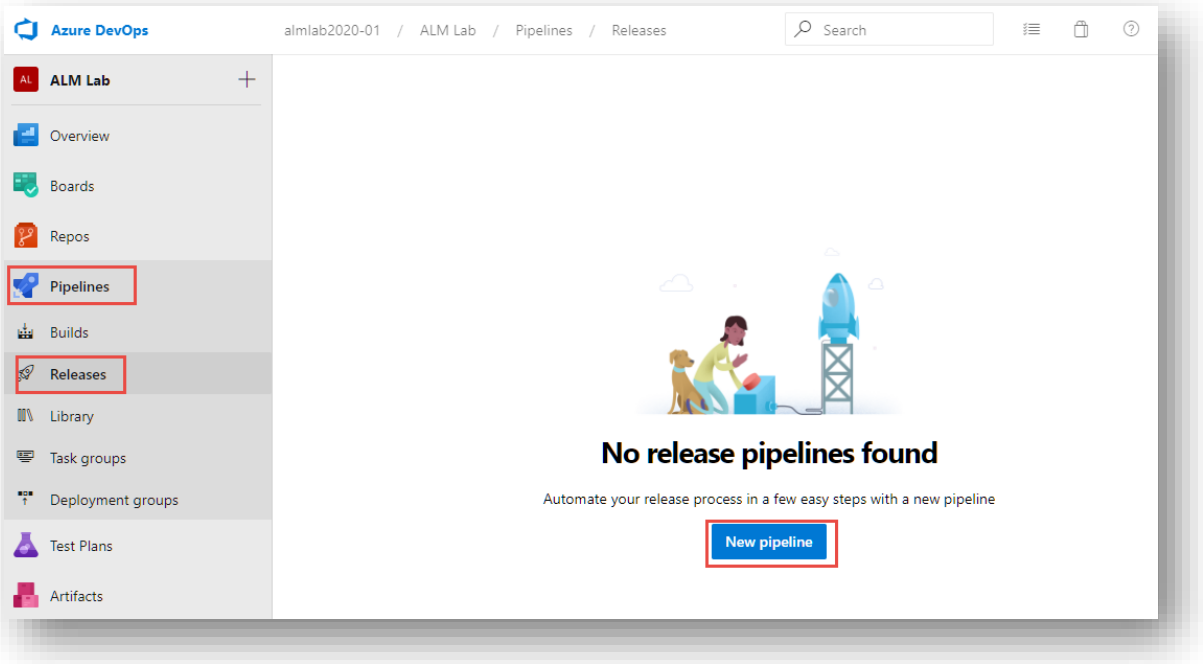
54. Edit the Unpack task and change the solution type to Both



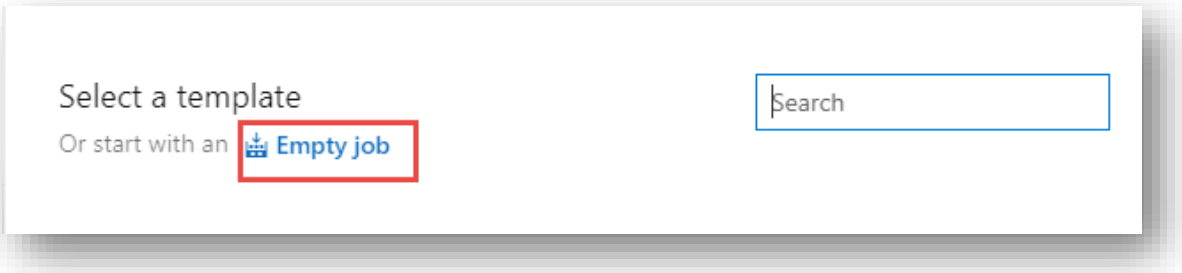
55. Save, Queue, Run, and inspect your results of the pipeline.

Create Release Pipeline – deploy to Production

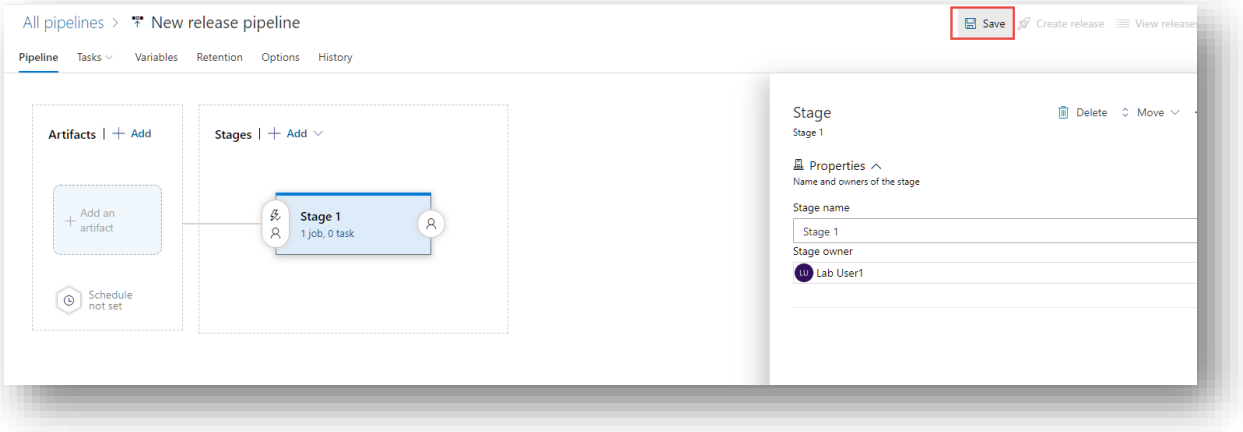
56. Now it is time to set up your release pipeline. In order to import one, we need to create a blank one first. Select Pipelines, Releases, and Click New pipeline



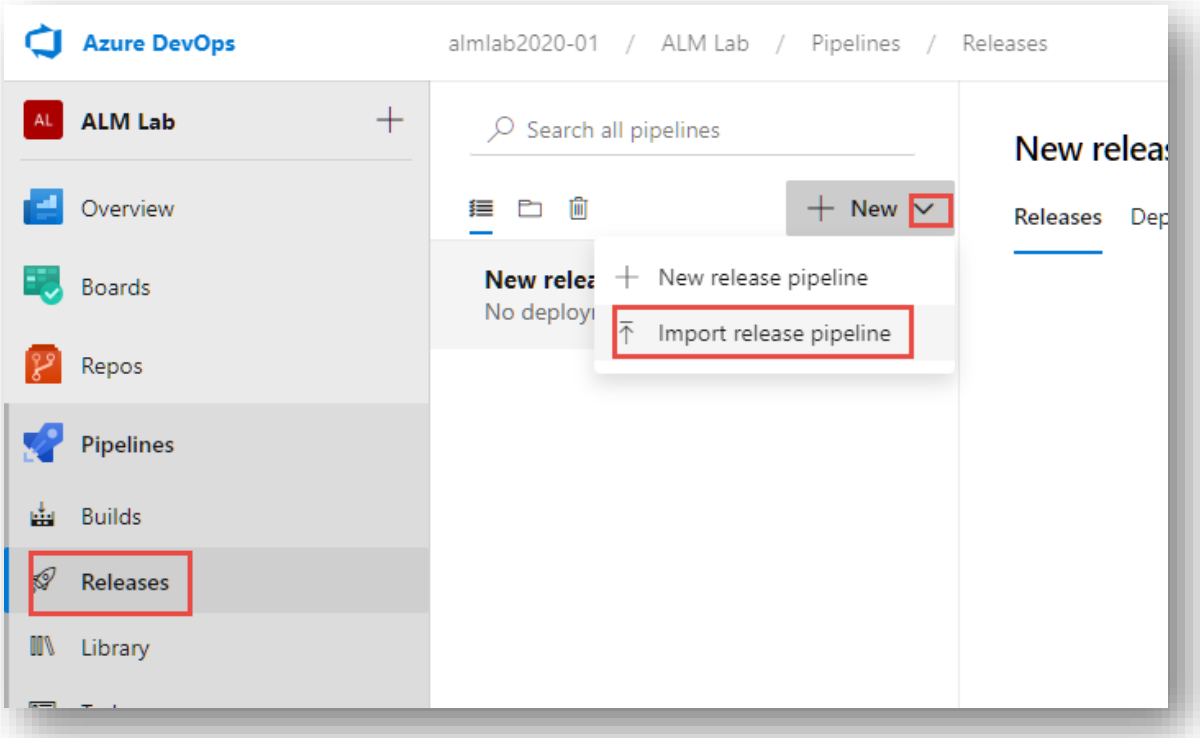
57. Select Empty job



58. Click Save



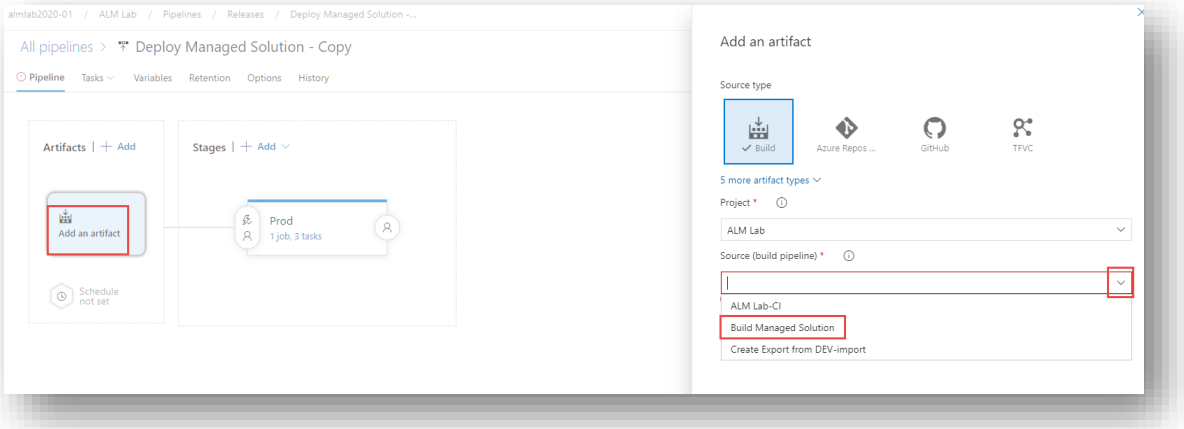
59. Go back to your release pipeline list and click New and Import release pipeline option from the dropdown.



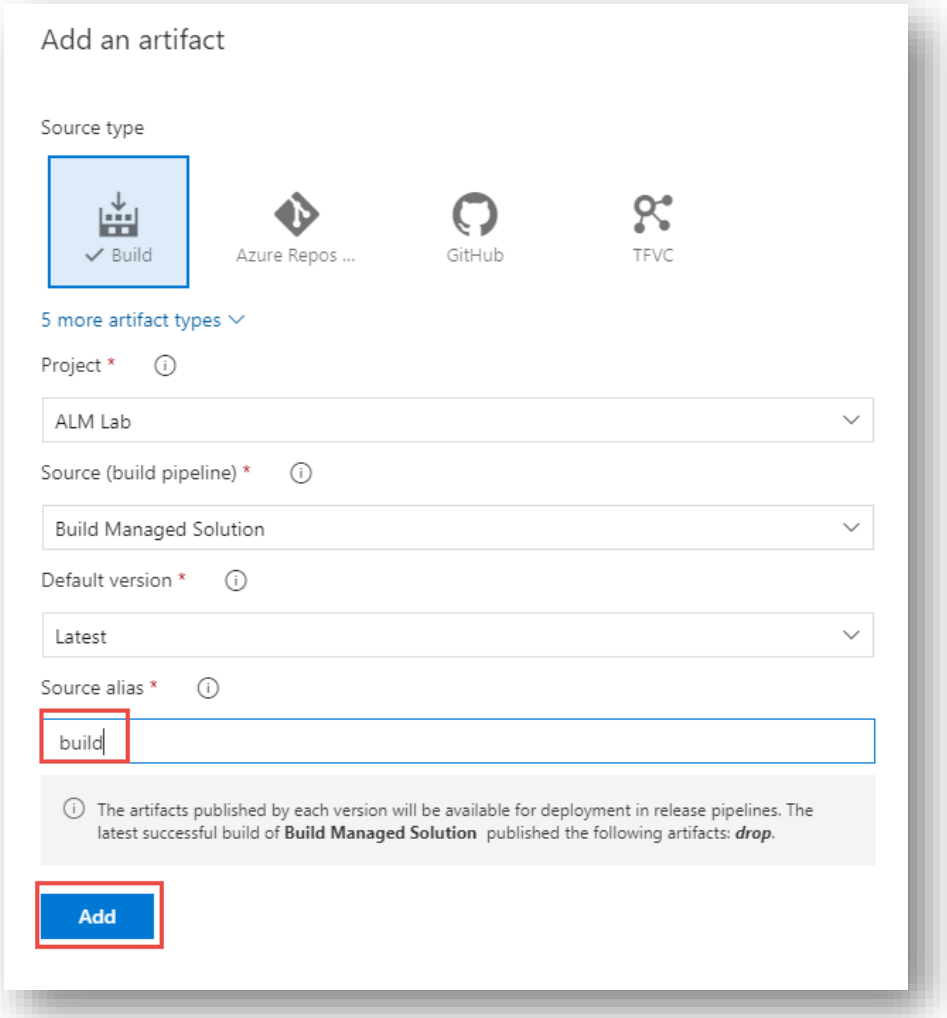
60. Browse to and select the Build Managed Pipeline.json pipeline provided in the lab materials. Change the SolutionName variable to ProductList.

61. Browse to and select the Deploy Managed Solution.json pipeline provided in the lab materials and click OK.

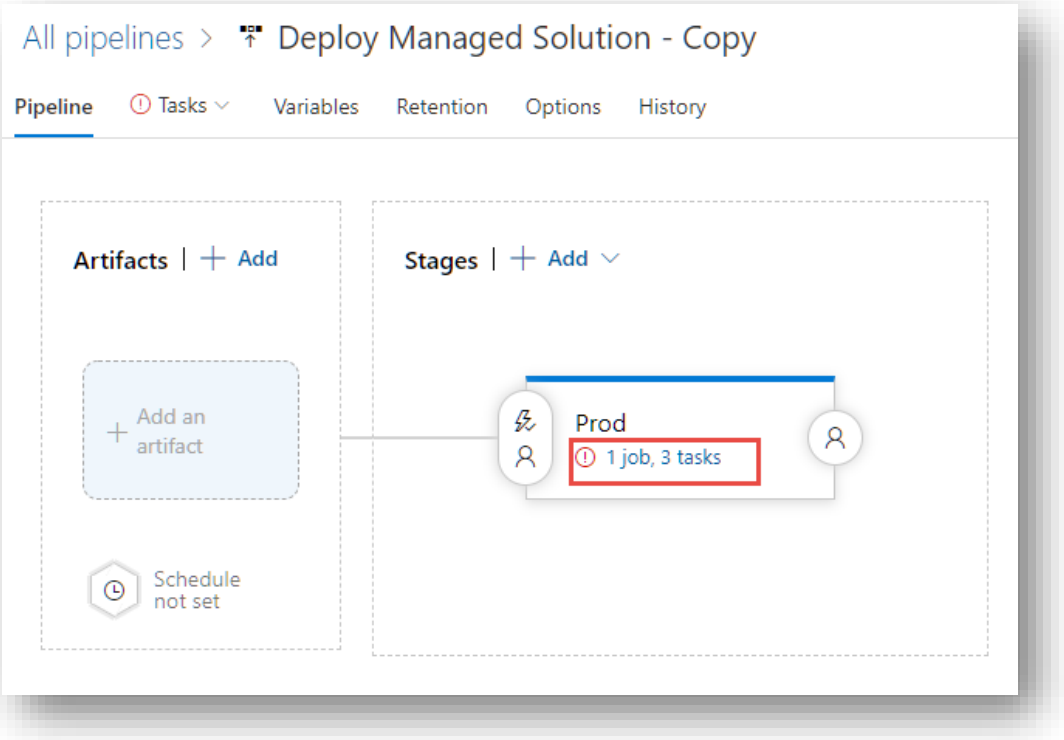
62. Click the Add an artifact area and select the source pipeline, which is your Build Managed Solution pipeline.



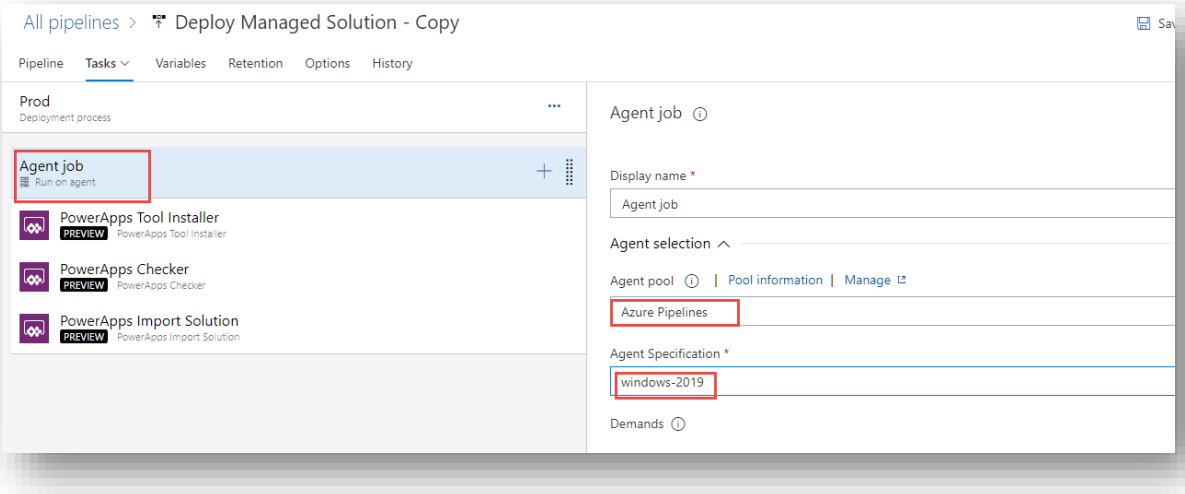
63. Enter build for the pipeline name and click Add



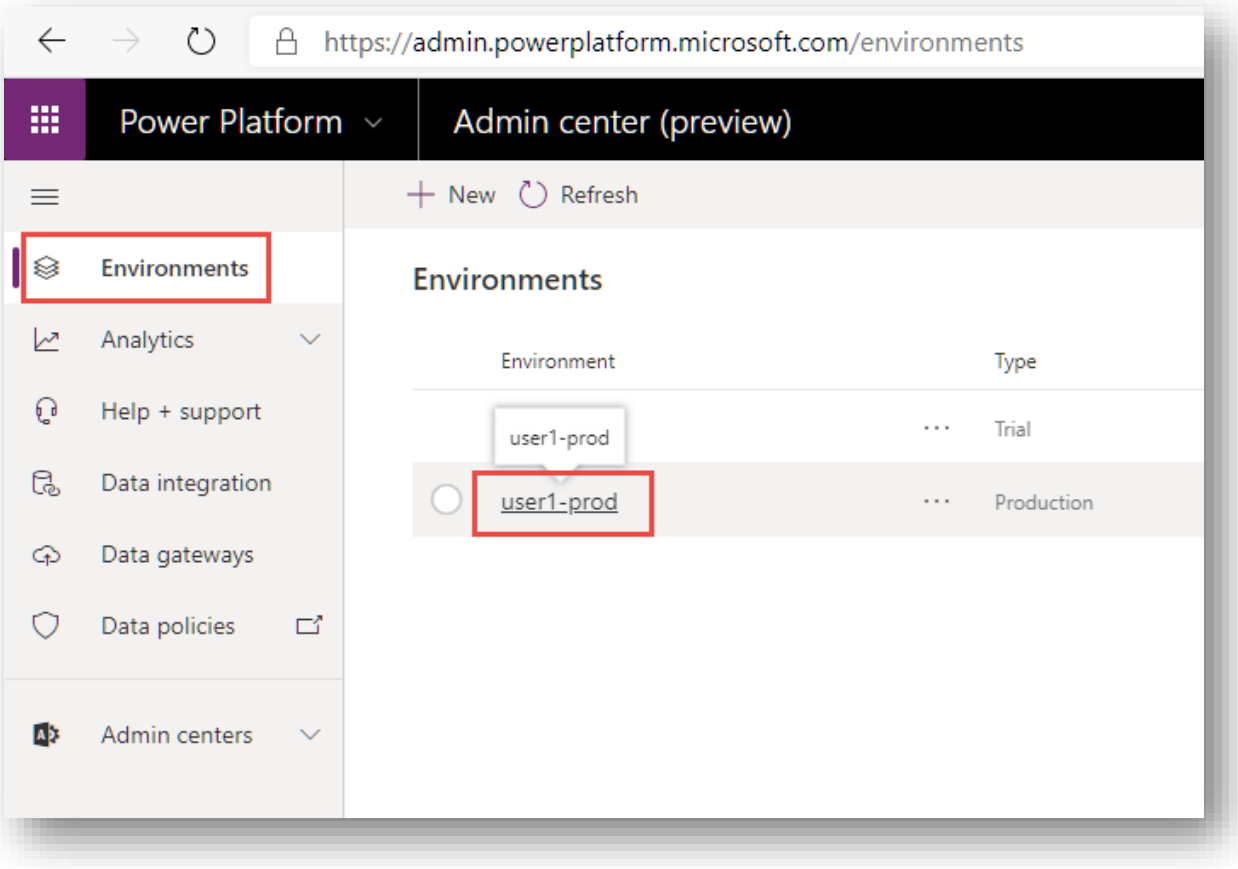
64. Click the Job and tasks link to edit the pipeline



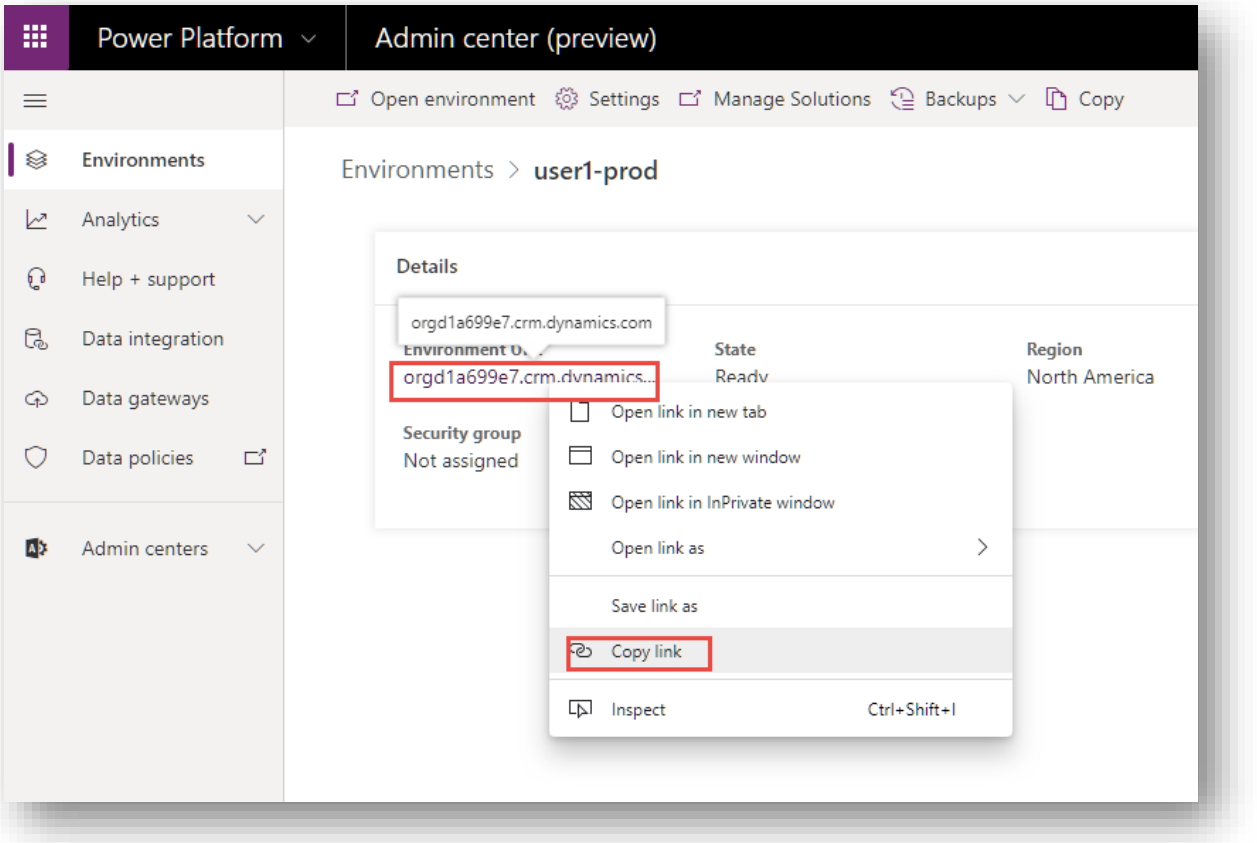
65. Click the Agent Job panel, then set the agent settings



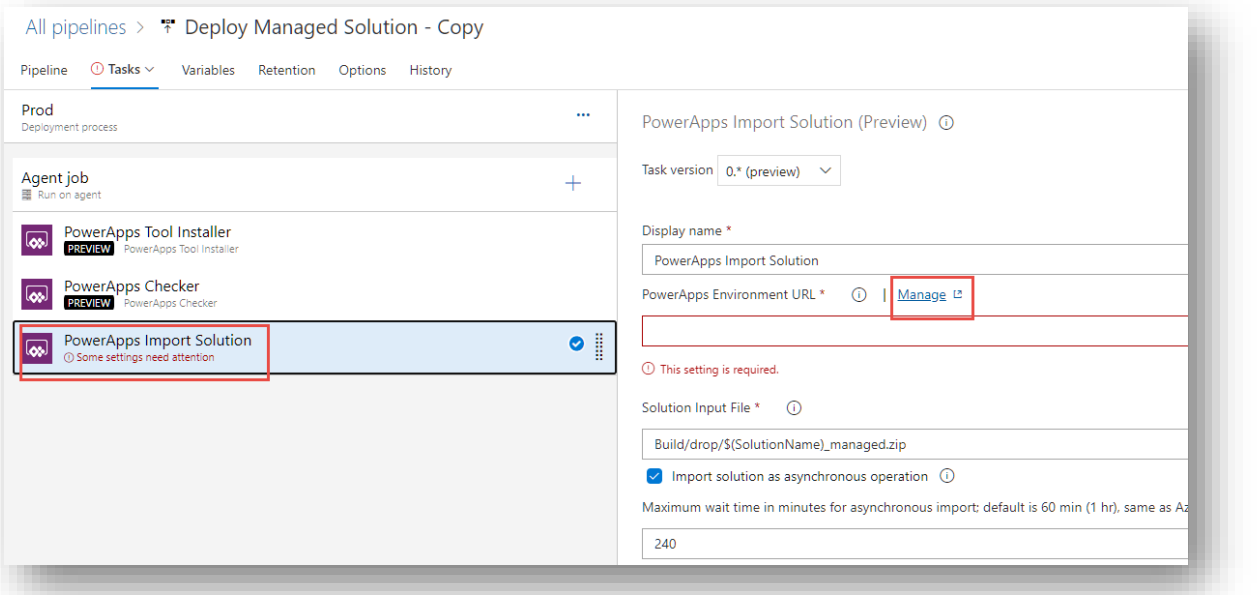
66. Switch back to your Power Platform Admin tab and click on your production environment



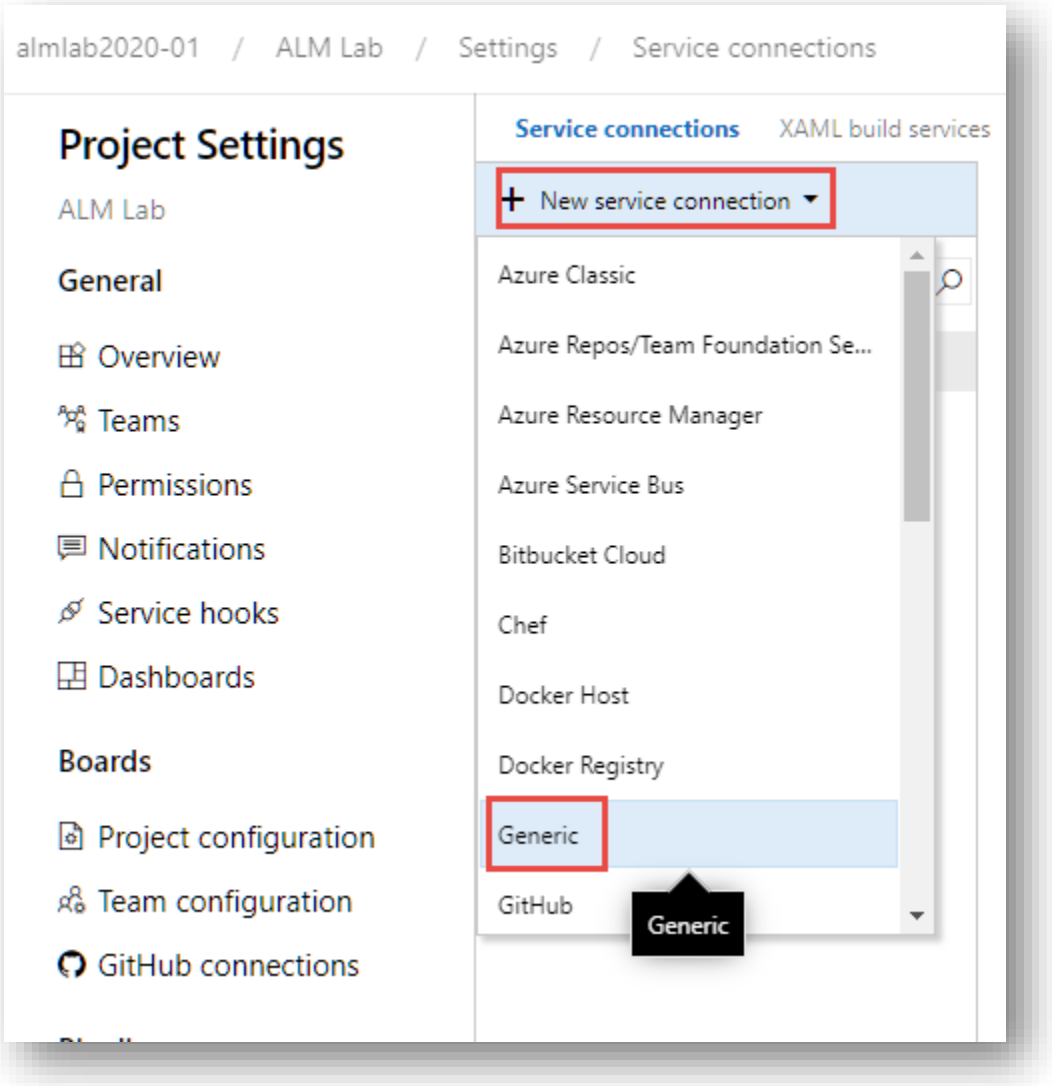
67. Right-click on the Environment URL and copy it to your clipboard



68. Switch back to your pipeline and select the Import Solution task, and click the Manage button on the environment Url.

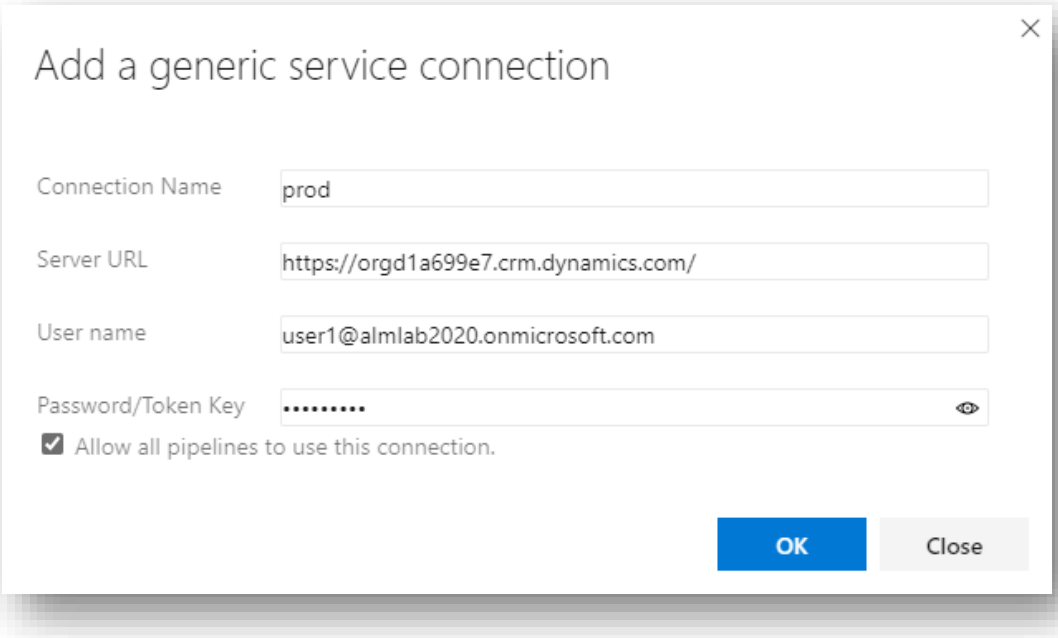


69. Create a new service connection and select Generic as the type

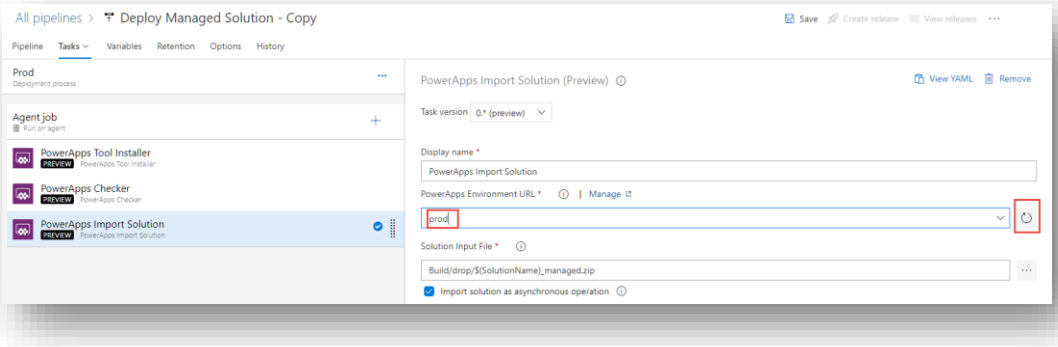


70. Enter a name for your production environment, paste the environment Url you have in your clipboard, and enter the credentials supplied in your lab and click OK to create your service

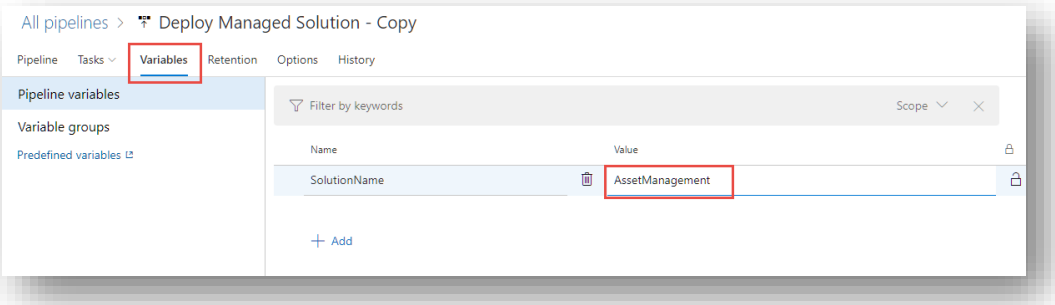
connection.



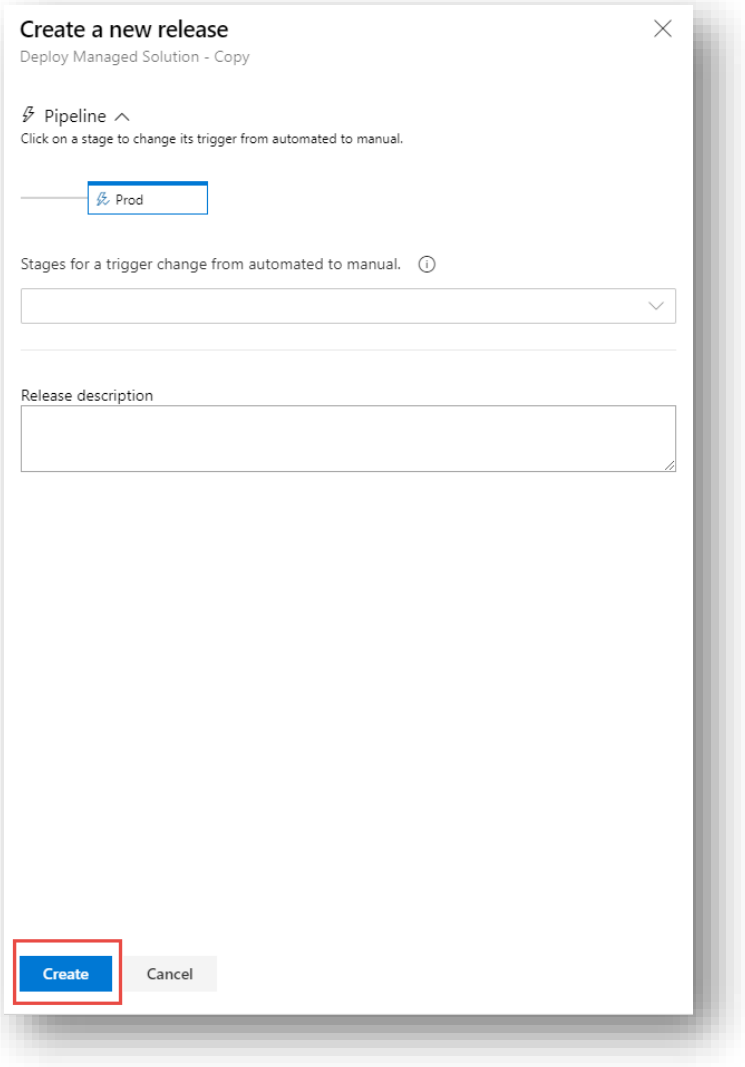
71. Back on your pipeline tab, select the new connection you just created. If it doesn't exist yet, click the refresh icon to make it show up.



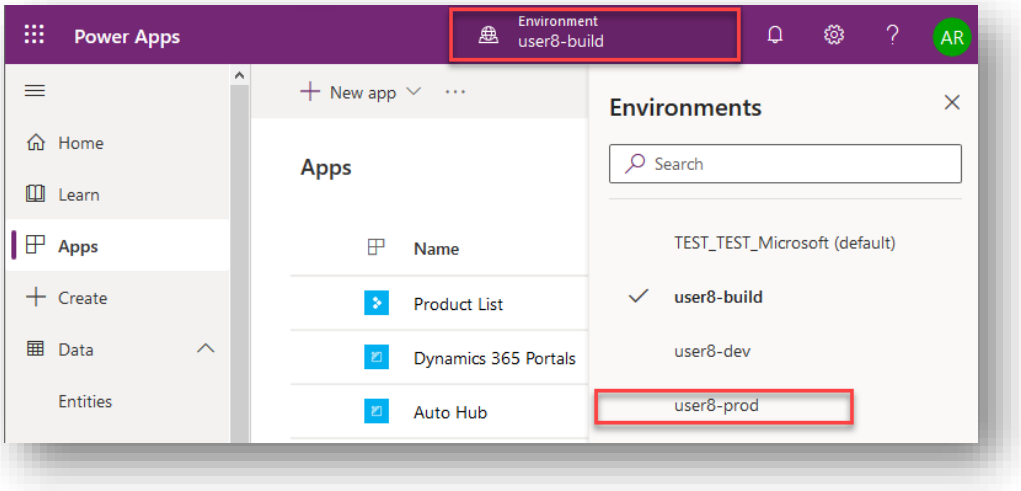
72. Click the Variables tab and set the SolutionName value to ProductList, then Save the pipeline.



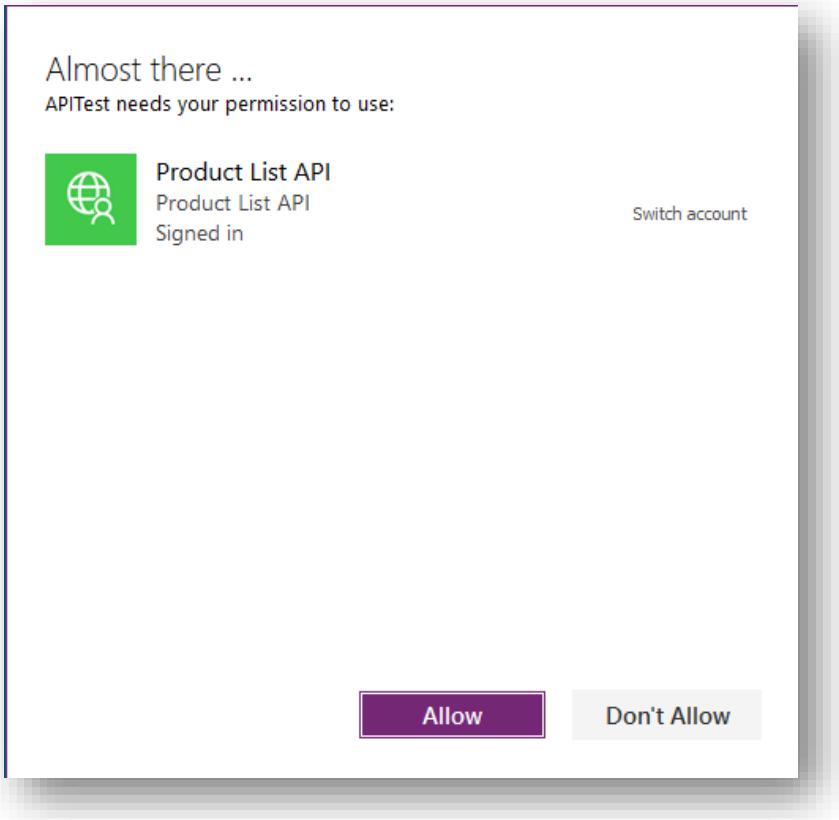
73. Click the Create Release button and click Create



74. Go back to make.powerapps.com and select the production environment that the Pipeline deployed the App to. Validate that the solution was successfully imported and play the app in the new environment.



75. First time you play the App in the new environment, you will be prompted to allow the App to use the Connector. Click **Allow** and confirm that the app works as expected in the production environment.



Lab Disclaimer

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

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

The names of manufacturers, products, or URLs are provided for informational purposes only and Microsoft makes no representations and warranties, either expressed, implied, or statutory, regarding these manufacturers or the use of the products with any Microsoft technologies. The inclusion of a manufacturer or product does not imply endorsement of Microsoft of the manufacturer or product. Links may be provided to third party sites. Such sites are not under the control of Microsoft and Microsoft is not responsible for the contents of any linked site or any link contained in a linked site, or any changes or updates to such sites. Microsoft is not responsible for webcasting or any other form of transmission received from any linked site. Microsoft is providing these links to you only as a convenience, and the inclusion of any link does not imply endorsement of Microsoft of the site or the products contained therein.

Microsoft and the trademarks listed at https://www.microsoft.com/en-us/legal/intellectualproperty/Trademarks/Usage/General.aspx are trademarks of the Microsoft group of companies. All other trademarks are property of their respective owners.