Contents

1 PL-400 Microsoft Power Platform Developer

- **Download Latest Student Handbook and AllFiles Content**
- **Are you a MCT?** - Have a look at our GitHub User Guide for MCTs
- **Need to manually build the lab instructions?** - Instructions are available in the MicrosoftLearning/Docker-Build repository

1.1 What are we doing?

- To support this course, we will need to make frequent updates to the course content to keep it current with the Azure services used in the course. We are publishing the lab instructions and lab files on GitHub to allow for open contributions between the course authors and MCTs to keep the content current with changes in the Azure platform.
- We hope that this brings a sense of collaboration to the labs like we've never had before - when Azure changes and you find it first during a live delivery, go ahead and make an enhancement right in the lab source. Help your fellow MCTs.

1.2 How should I use these files relative to the released MOC files?

- The instructor handbook and PowerPoints are still going to be your primary source for teaching the course content.
- These files on GitHub are designed to be used in conjunction with the student handbook, but are in GitHub as a central repository so MCTs and course authors can have a shared source for the latest lab files.
- It will be recommended that for every delivery, trainers check GitHub for any changes that may have been made to support the latest Azure services, and get the latest files for their delivery.

1.3 What about changes to the student handbook?

- We will review the student handbook on a quarterly basis and update through the normal MOC release channels as needed.

1.4 How do I contribute?

- Any MCT can submit a pull request to the code or content in the GitHub repro, Microsoft and the course author will triage and include content and lab code changes as needed.
- You can submit bugs, changes, improvement and ideas. Find a new Azure feature before we have? Submit a new demo!

1.5 Notes

1.5.1 Classroom Materials

1.6 It is strongly recommended that MCTs and Partners access these materials and in turn, provide them separately to students. Pointing students directly to GitHub to access Lab steps as part of an ongoing class will require them to access yet another UI as part of the course, contributing to a confusing experience for the student. An explanation to the student regarding why they are receiving separate Lab instructions can highlight the nature of an always-changing cloud-based interface and platform. Microsoft Learning support for accessing files on GitHub and support for navigation of the GitHub site is limited to MCTs teaching this course only.

1.7 title: Online Hosted Instructions permalink: index.html layout: home

2 Content Directory

Hyperlinks to each of the lab exercises and demos are listed below.

2.1 Labs

{% assign labs = site.pages | where_exp:"page", "page.url contains '/Instructions/Labs'" %} | Lab | | --- | {% for activity in labs %} | [{{ activity.lab.title }}{% if activity.lab.type %} - {{ activity.lab.type }}{% endif %}](/home/ll/Azure_clone/Azure_new/PL-400_Microsoft-Power-Platform-Developer/{{ site.github.url }}{{ activity.url }}) | {% endfor %}

2.2 lab: title: 'Lab 00: Validate lab environment'

[!NOTE] Effective November 2020:

- Common Data Service has been renamed to Microsoft Dataverse. Learn more
- Some terminology in Microsoft Dataverse has been updated. For example, *entity* is now *table* and *field* is now *column*. Learn more

This content will be updated soon to reflect the latest terminology.

2.3 Practice Lab – Validate lab environment

Attention to MCTs: Please make sure you are familiar with the TrainerPrepGuide for this course, especially the teaching tips and recommendations.

2.4 Scenario

In this Module 0 lab, you will acquire a Power Platform trial tenant, access the Power Platform admin center and setup your Azure DevOps account. In the admin center, we will create an individual environment for configuration during the course.

2.5 Exercise 1 – Acquire your Power Platform trial tenant

1. Copy your **Microsoft 365 credentials** from the Authorized Lab Hoster.
2. Navigate to <powerapps.microsoft.com> and click **Start free.**
3. Under **Work email**, enter the email address from your Microsoft 365 credentials.
4. You see a prompt that you have an existing account with Microsoft. Select **Sign in.**
5. Enter the password provided by the Authorized Lab Hoster.
6. Select **Yes** to stay signed in.

2.6 Exercise 2 - Create your environment

In this exercise, you will create your **Development** environment that you will do the majority of your lab work in.

2.6.1 Task 1 – Create environment

1. Access https://admin.Powerplatform.microsoft.com and log in with your Microsoft 365 credentials if prompted again.
2. Select **Environments** and click **+New.**
   - For **Name**, enter **[my initials] Dev** (Example: AJ Dev)
   - For **Type**, select **Trial.**
   - Change the toggle on **Create a database for this environment?** to **Yes.**
   - Leave all other selections as default and click **Next.**
   - On the next tab, leave all selections to default and click **Save**
3. Your **Dev** environment should now show in the list of Environments.

4. Your environment may take a few minutes to provision. Refresh the page if needed. When your environment is prepared, select your **Dev** environment by clicking on the ellipses next to its name to expand the drop down menu and select **Settings.**
5. Explore the different areas in **Settings** that you are interested in but do not make any changes yet.

2.7 Exercise 3 - Azure DevOps account setup

In this exercise, you will create your Azure DevOps account that you will be using in Lab10

1. Get a new Azure Pass (valid for 30-days) from the instructor or other source.
2. Use a private browser session, go to Microsoftazurepass.com to redeem your Azure Pass using the Dynamics 365 (or M365) credentials provided to you). Redeem a Microsoft Azure Pass Follow the instructions for redemption.
3. Using the same browser session, go to portal.azure.com, then search for "Azure DevOps". In the resulting page, click Azure DevOps Organizations.
4. Next, click on the link called "My Azure DevOps Organizations" (or navigate to https://aex.dev.azure.com/).
5. In the drop down box on the left, choose Default, instead of "Microsoft Account"
6. Create a new organization (find blue box in upper right-hand corner of the screen) using the Default directory. Provide a unique Azure DevOps Organization name like FL- PermitManagement (replace FL with your first and last initials), select your region and click Continue.
7. Choose the newly created organization, then choose Organization settings on the left-hand side of the screen
8. Navigate to Organization settings -> Billing -> Setup billing -> Select an Azure subscription, then select the Azure Pass subscription, then choose "MS Hosted CI/CD" and set the field "Paid parallel jobs" to 1. Then click SAVE in the blue box at the bottom.
9. Wait at least 3 hours before using the CI/CD capabilities so that new settings are reflected in the back end. Otherwise you will still see the message "This agent is not running because you have reached the maximum number of requests…".
10. As an optional step, you can validate this by creating a new pre-defined project using the newly created org with billing enabled, using https://azuredevopsdemogenerator.azurewebsites.net/. Wait for some time before trying, then run a test build.

2.8 lab: title: 'Lab 01: Data Modeling'

[!NOTE] Effective November 2020:

- Common Data Service has been renamed to Microsoft Dataverse. Learn more
- Some terminology in Microsoft Dataverse has been updated. For example, *entity* is now *table* and *field* is now *column*. Learn more

This content will be updated soon to reflect the latest terminology.

2.9 Lab 01 – Data Modeling

3 Scenario

A regional building department issues and tracks permits for new buildings and updates for remodeling of existing buildings. Throughout this course you will build applications and perform automation to enable the regional building department to manage the permitting process. This will be an end-to-end solution which will help you understand the overall process flow.

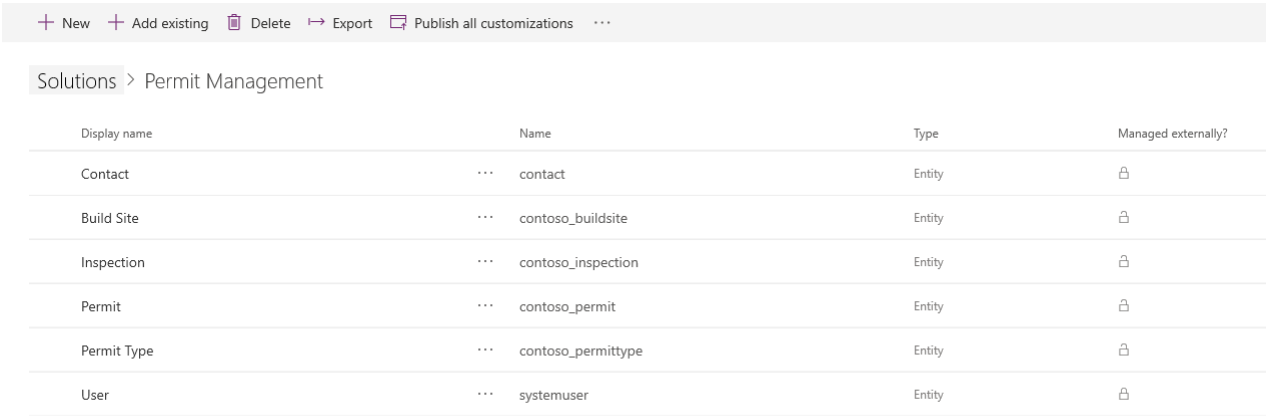
In this lab, you will set up a second environement to memic a production environment for learning purposes, create solutions to track your changes. You will also create a data model to support the following requirements:

- R1 – Track the status of permits issued for new buildings and existing building modifications
- R2 – Permits are associated with a Build Site, which represents the building or land being modified

- R3 – Permit type indicates the type of permit and inspections, other data that might be required on a permit
- R4 – Inspections completed on the permit work are to be tracked for the entire process i.e., from request of inspection to the pass or fail of the inspection
- R5 – Permits, for our lab purposes, are requested by a person and we need to track who requested each permit

4 High-level lab steps

To prepare your learning environments you will create a solution and publisher and add both new and existing components that are necessary to meet the application requirements. Refer to the data model document for the metadata description (Tables, Column types and relationships). Your solution will contain several Tables upon completion of all the customizations.



4.1 Things to consider before you begin

- What are considered as best practices for managing changes in between environments (“Dev” to “Test” to “Prod”)? Are there additional considerations for team solution development?
- What Tables a user might need in the scenario that we are building?
- What relationship behaviors would we consider enabling users to complete their tasks?
- Remember to work in your **DEVELOPMENT** environment with the customizations. Once the customizations are completed, published and tested in “Dev”, and if everything works fine, the same will be deployed to “Prod”.

5 Exercise #1: Create Environments and Solution

Objective: In this exercise, you will create a community plan environment to memic Production environment that we will refer to as ”Prod”.

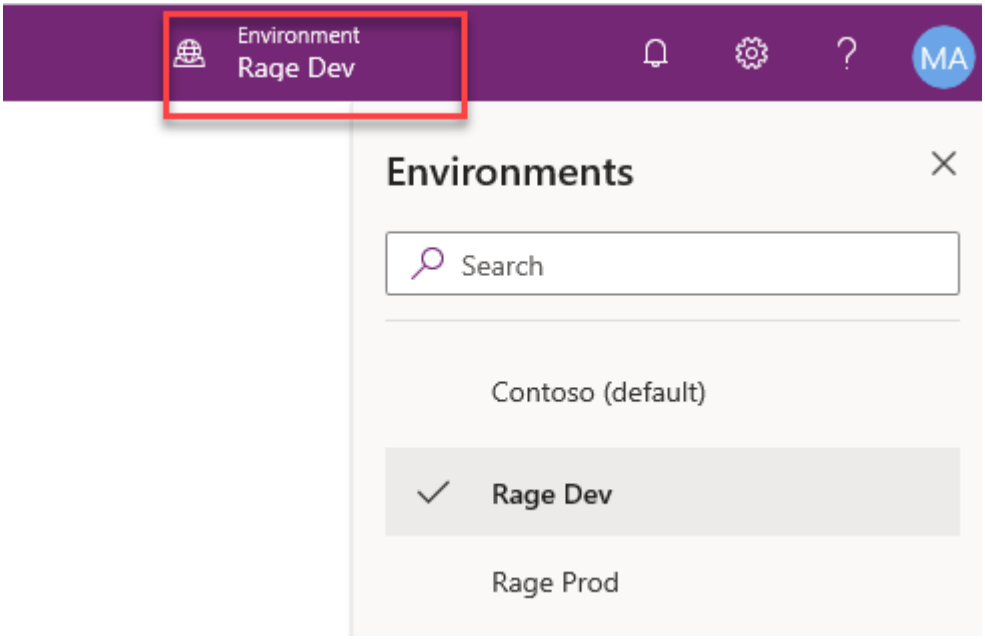
5.1 Task #1: Create Environments

1. Create the community plan environment
   - Navigate to Power Apps Community Plan page
   - Click on *Create an indiviual environment*
   - Enter your credentials when prompt to sign in
   - Select your country from the dropdown menu and click *Accept*
   - Navigate to Power Platform Admin Center to see a new environment had been created by the system. We will refer to it as ”Prod” environment for the rest of this course.

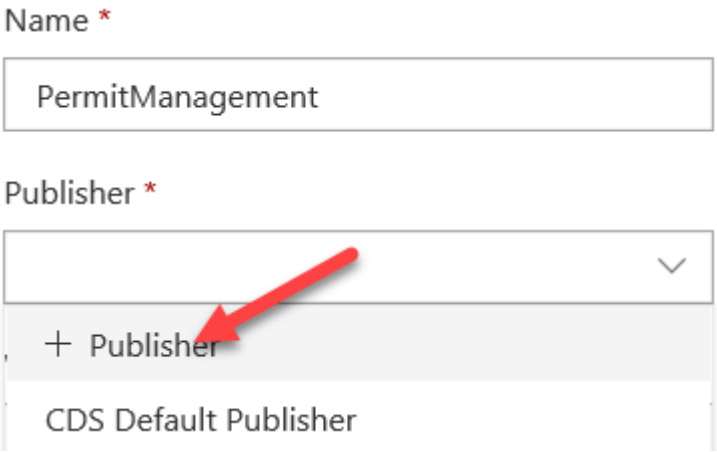
You should now have the dev environment and the ”Prod” environment listed under environments.

5.2 Task #2: Create Solution and Publisher

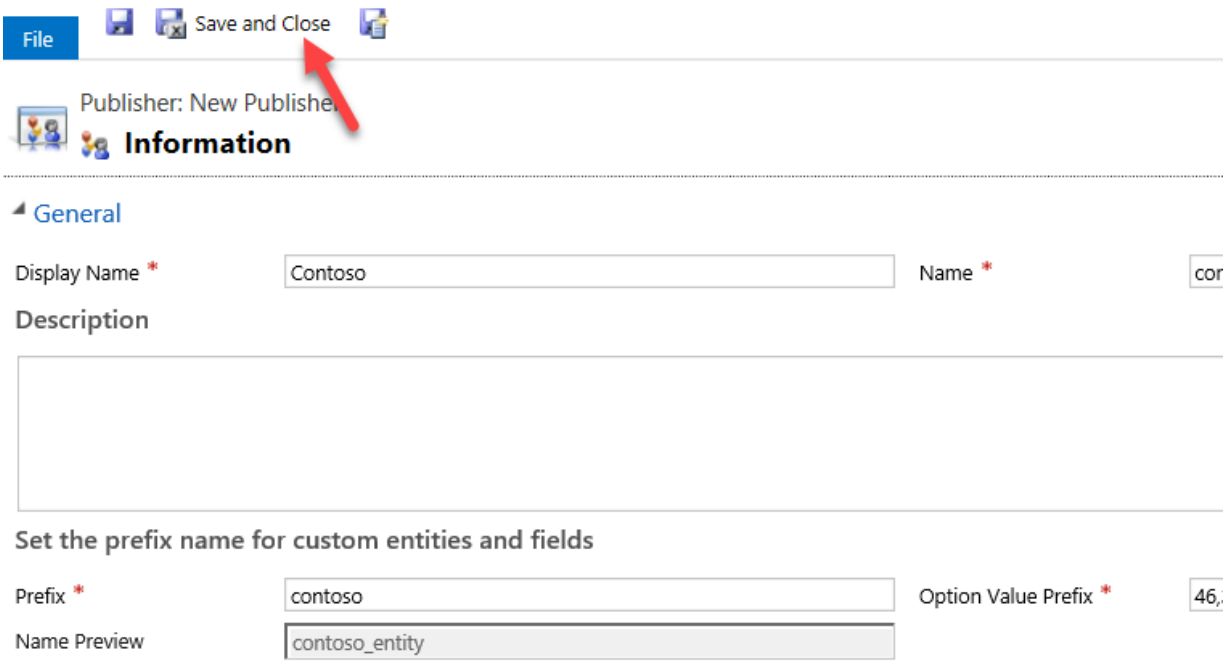
1. Create Solution
   - Sign in to Power Apps maker portal
   - Select your Dev environment.



   - Select **Solutions** from the left menu and click **+ New solution**.
   - Enter **Permit Management** for **Display Name**.
2. Create Publisher
   - Click on the **Publisher** dropdown and select **+ Publisher**.



   - Enter **Contoso** for **Display Name** and **contoso** for **Prefix.**
   - Click **Save and Close**.

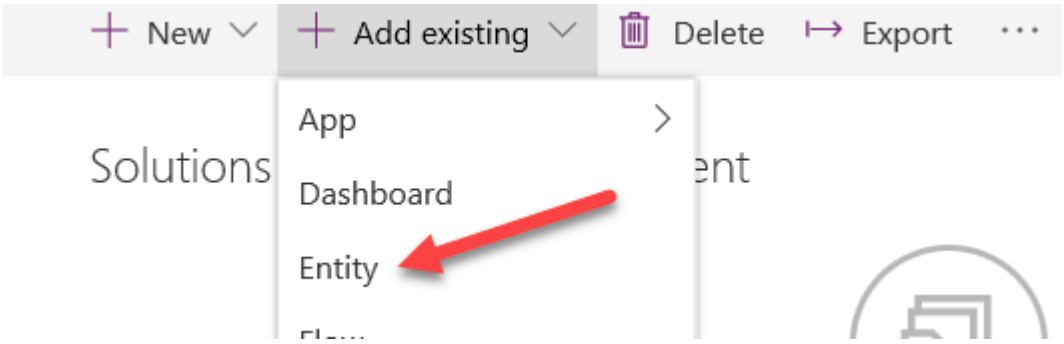


- Click **Done**.

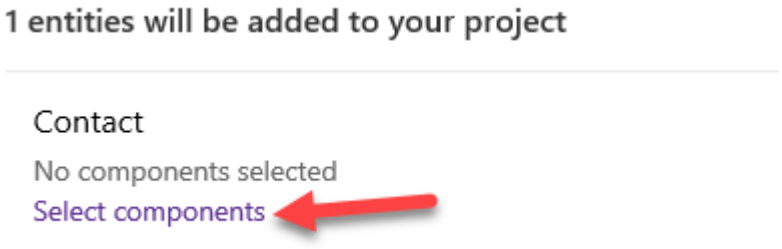
3. Complete the solution creation
   - Now, click on the **Publisher** dropdown and select the **Contoso** publisher you just created.
   - Enter **1.0.0.0** for **Version** and click **Create**.

5.3 Task #3: Add Existing Table

1. Add Contact Table to the solution
   - Click to open the **Permit Management** solution you just created.
   - Click **Add Existing** and select **Table**.

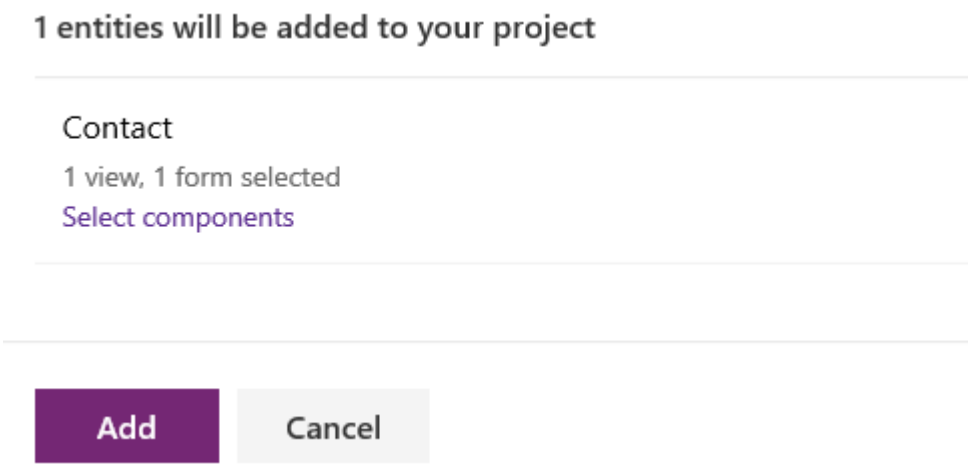


   - Search for **Contact** and select it.
   - Click **Next**.
   - Click **Select Components**.



   - Select the **Views** tab and select the **Active Contacts** view. Click **Add**.

- Click **Select Components**.
- Select the **Forms** tab and select the **Contact** form.
- Click **Add**.
- You should have **1 View** and **1 Form** selected. Click **Add** again. This will add the Contact Table to the newly created solution.



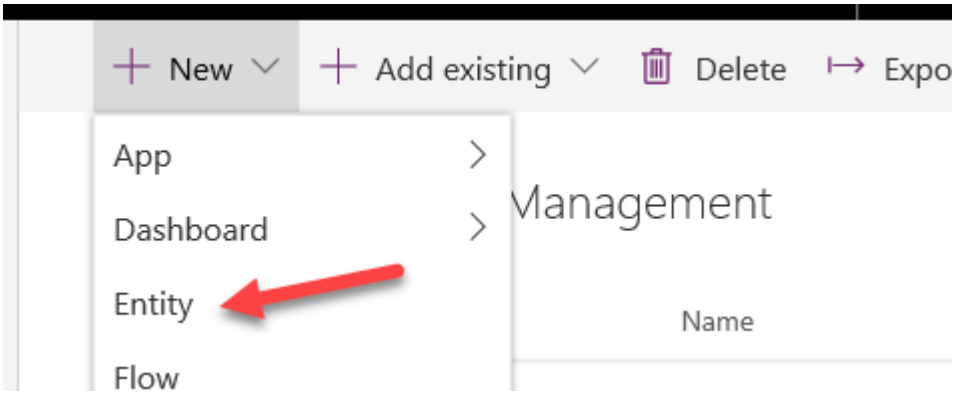
2. Add User Table to the solution
   - Click Add Existing and select **Table**.
   - Search for **User** and select it.
   - Click **Next**.
   - **DO NOT** select any components. Click **Add**.
   - Your solution should now have two Tables.

6 Exercise #2: Create Tables and Columns

Objective: In this exercise, you will create Tables, add Columns to these Tables and edit the **Status Reason** options for the **Permit** and **Inspection** Tables.

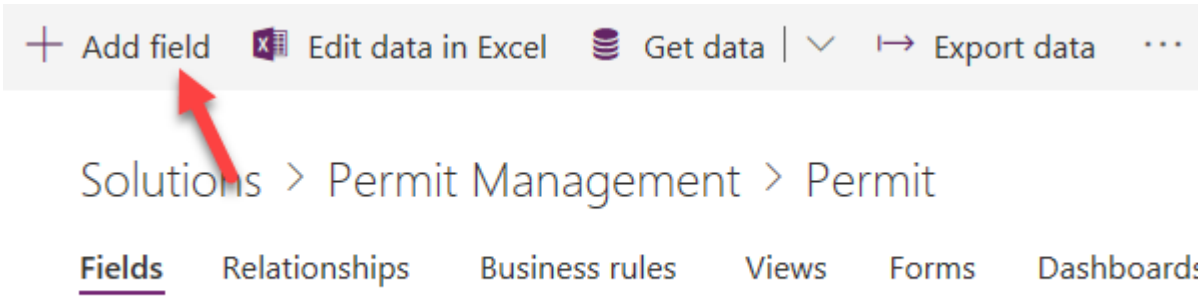
6.1 Task #1: Create Permit Table and Columns

1. Continuing in your development environment, open the Permit Management solution
   - Sign in to Power Apps maker portal
   - Select **Solutions** and click to open the **Permit Management** solution you just created.
2. Create Permit Table
   - Click + **New** and select **Table**.

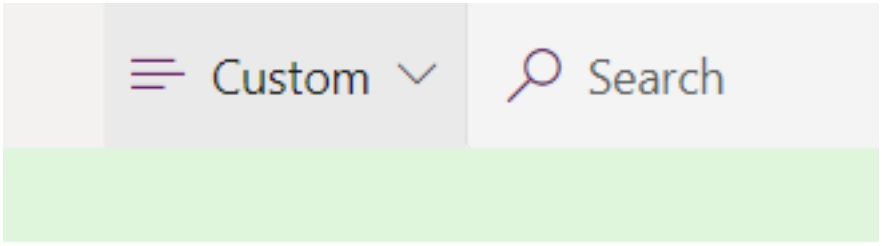


- Enter **Permit** for **Display Name** and click **Create**. This will start provisioning the Table in background while you can start adding Columns.

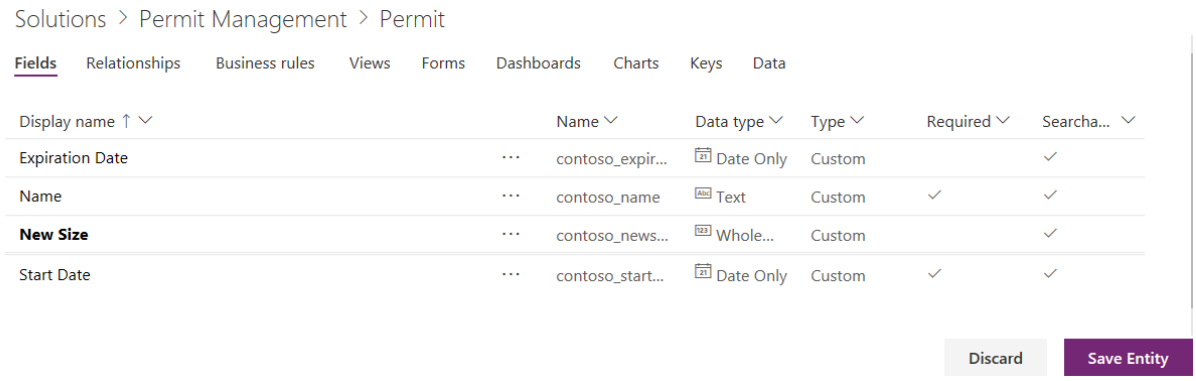
3. Create Start Date Column
   - Make sure you have the **Columns** tab selected and click + **Add Column**.



   - Enter **Start Date** for **Display Name**.
   - Select **Date Only** for **Data Type**.
   - Select **Required**.
   - Leave the searchable checkbox checked. When a Column is searchable it appears in Advanced Find in model-driven apps and is available when customizing views. De-selecting this will reduce the number of options shown to people using advanced find.
   - Click **Done**.
4. Create Expiration Date Column.
   - Click + **Add Column**.
   - Enter **Expiration Date** for **Display Name**.
   - Select **Date Only** for **Data Type**.
   - Click **Done**.
5. Create New Size Column.
   - Click + **Add Column**.
   - Enter **New Size** for **Display Name**.
   - Select **Whole Number** for **Data Type.**
   - Click **Done**.
   - Select **Custom** for filter.

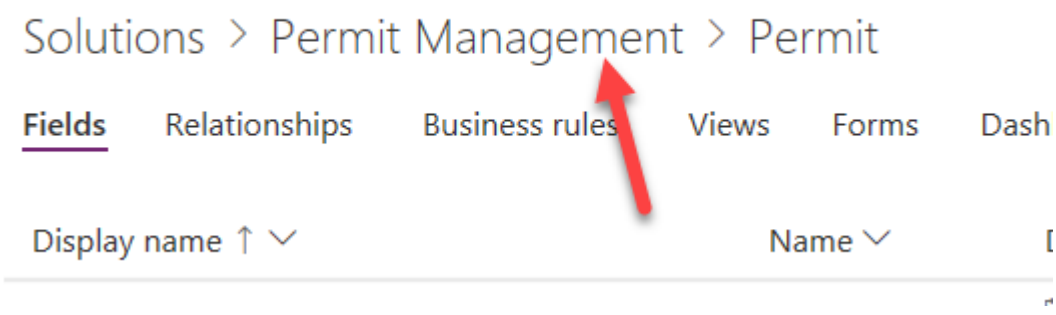


   - Click **Save Table**.



6.2 Task #2: Create Permit Type Table and Columns

1. Create Permit Type Table
   - Click on the solution name. This action will take you back to the Solution.

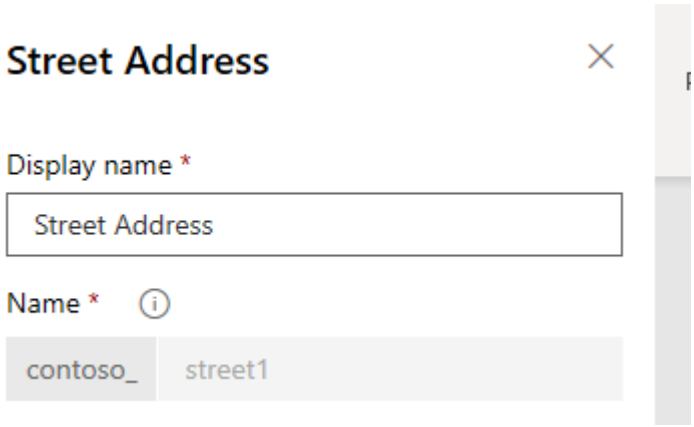


   - Click + **New** and select **Table**.
   - Enter **Permit Type** for **Display Name**.
   - Click **Done**.
2. Create Require Inspections Column
   - Make sure you have the **Columns** tab selected and click + **Add Column**.
   - Enter **Require Inspections** for **Display Name**.
   - Select **Yes/No** for **Data Type**.
   - Click **Done**.
3. Create Require Size Column
   - Click + **Add Column**.
   - Enter **Require Size** for **Display Name**.
   - Select **Yes/No** for **Data Type**.
   - Click **Done**.
4. Click Save Table

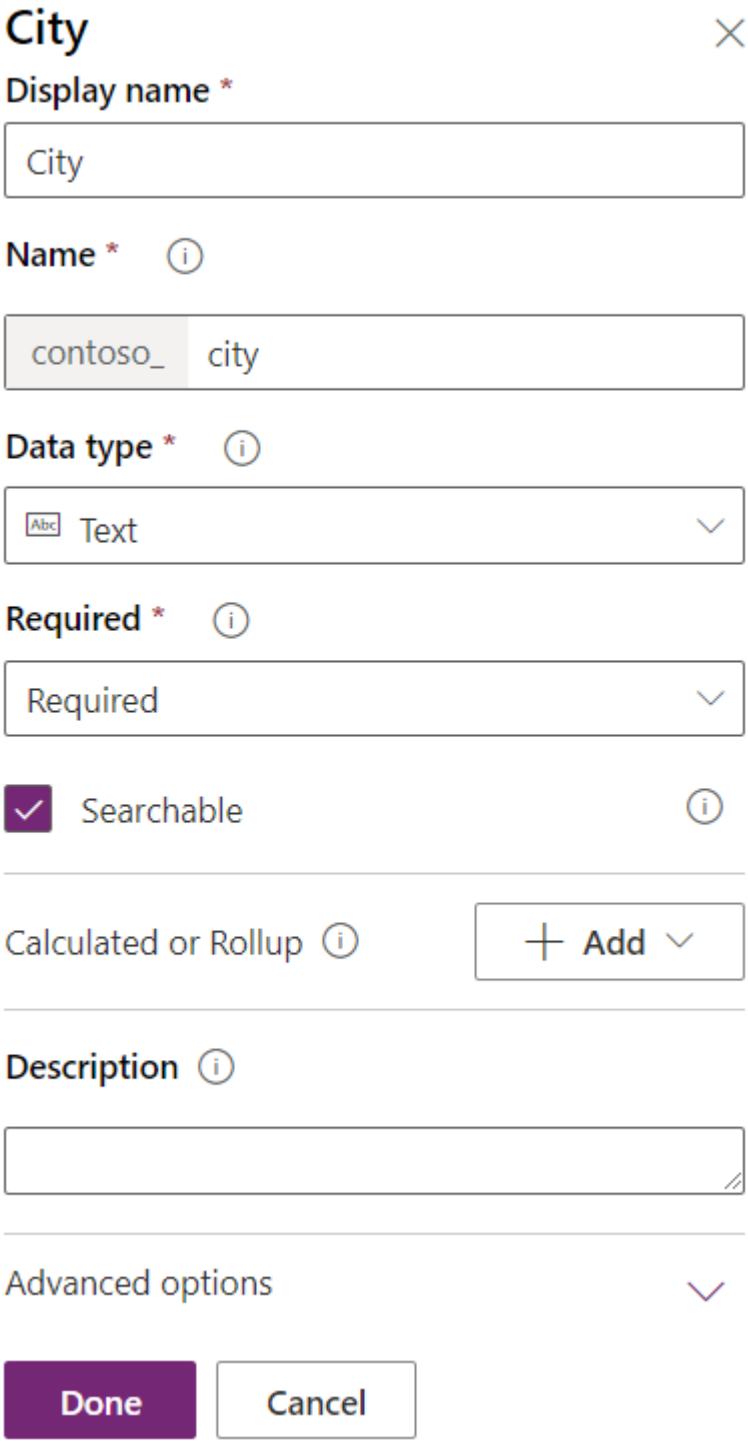
6.3 Task #3: Create Build Site Table and Columns

1. Create Build Site Table
   - Click on the solution name. This action will take you back to the Solution.
   - Click + **New** and select **Table**.
   - Enter **Build Site** for **Display Name.**
   - Change the **Display Name** of the **Primary Column** to **Street Address.**

- Change the **Name** of the **Primary Column** to **street1.**
- Click **Done**.



2. Add City Column
   - Make sure you have the **Columns** tab selected and click + **Add Column**.
   - Enter **City** for **Display Name** and change the **Name** to **city**.
   - Make sure **Text** is selected for **Data Type**.
   - Select **Required**.
   - Click **Done**.



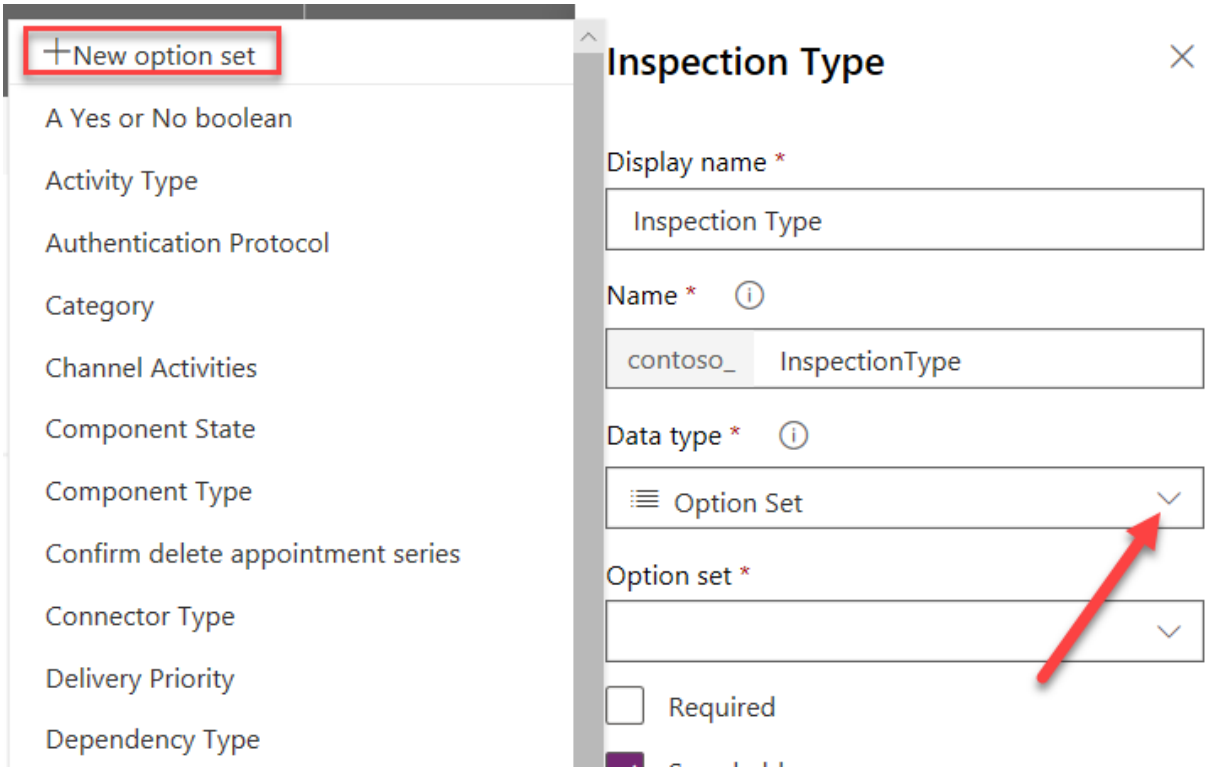
3. Add Zip/Postal Code Column
   - Make sure you have the **Columns** tab selected and click + **Add Column**.
   - Enter **ZIP/Postal Code** for **Display Name** and change the **Name** to **postalcode**.
   - Make sure **Text** is selected for **Data Type**.
   - Select **Required**.
   - Click **Done**.
4. Add State/Province Column
   - Make sure you have the **Columns** tab selected and click + **Add Column**.
   - Enter **State/Province** for **Display Name** and change the **Name** to **stateprovince**.

- Make sure **Text** is selected for **Data Type**.
- Select **Required**.
- Click **Done**.

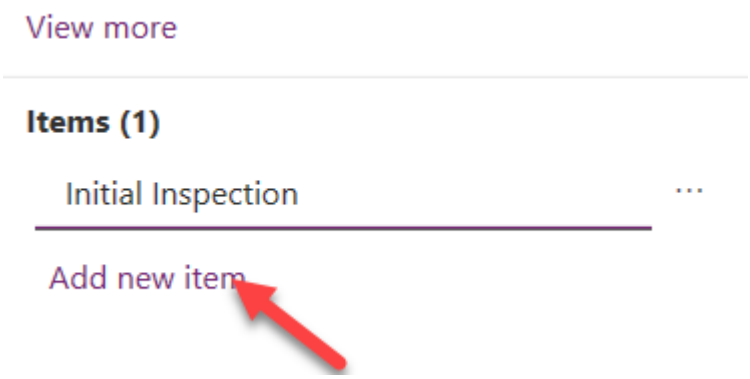
5. Add Country Region Column
   - Make sure you have the **Columns** tab selected and click + **Add Column**.
   - Enter **Country/Region** for **Display Name** and change the **Name** to **country**.
   - Make sure **Text** is selected for **Data Type.**
   - Click **Done**.
6. Click **Save Table**.

6.4 Task #4: Create Inspection Table and Columns

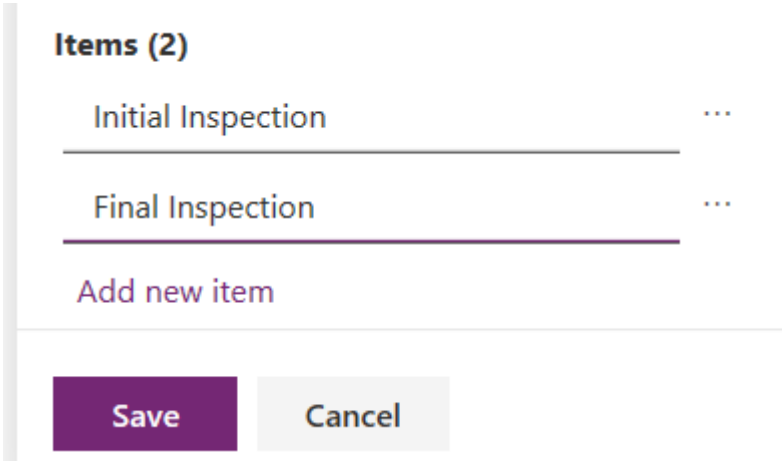
1. Create Inspection Table
   - Click on the solution name. This action will take you back to the Solution.
   - Click **New** and select **Table**.
   - Enter **Inspection** for **Display Name.**
   - Click **Done**.
2. Add Inspection Type Column
   - Make sure you have the **Columns** tab selected and click + **Add Column**.
   - Enter **Inspection Type** for **Display Name**.
   - Select **Choice** for **Data Type**.
   - Click on the **Choice** dropdown and select +**New Choice.**



- Enter **Initial Inspection** and click **Add New Item**.



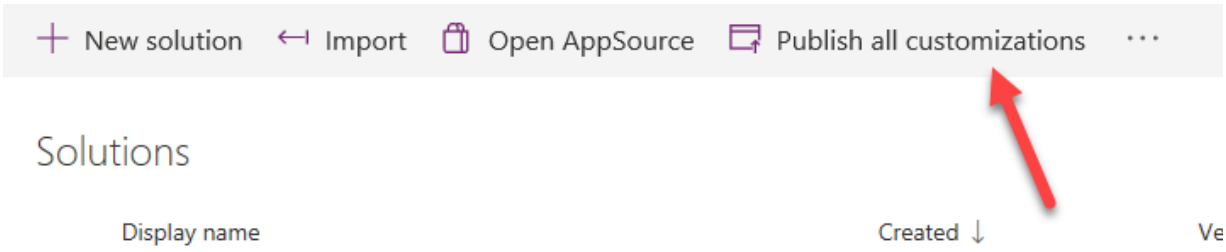
- Enter **Final Inspection** and click **Save**.



- Click **Done**.

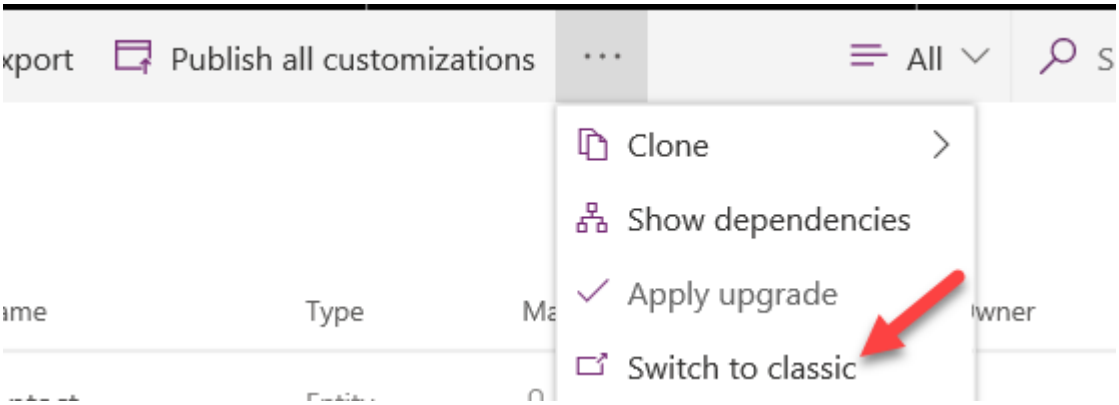
3. Add Scheduled Date Column
   - Make sure you have the **Columns** tab selected and click + **Add Column**.
   - Enter **Scheduled Date** for **Display Name**.
   - Select **Date Only** for **Data Type**.
   - Select **Required**.
   - Click **Done**.
4. Add Comments Column
   - Make sure you have the **Columns** tab selected and click + **Add Column**.
   - Enter **Comments** for **Display Name**.
   - Make sure **Text** is selected for **Data Type.**
   - Expand **Advanced options.**
   - Set **Max length to 1000** in the Advanced options**.**
   - Click **Done**.
5. Add Sequence Column
   - Make sure you have the **Columns** tab selected and click + **Add Column**.
   - Enter **Sequence** for **Display Name**.
   - Make sure **Text** is selected for **Data Type**.
   - Click **Done**.
6. Click **Save Table**.
7. Select **Solutions** on the top and this action will take you back to the Solutions page.

8. Click **Publish All Customizations.**



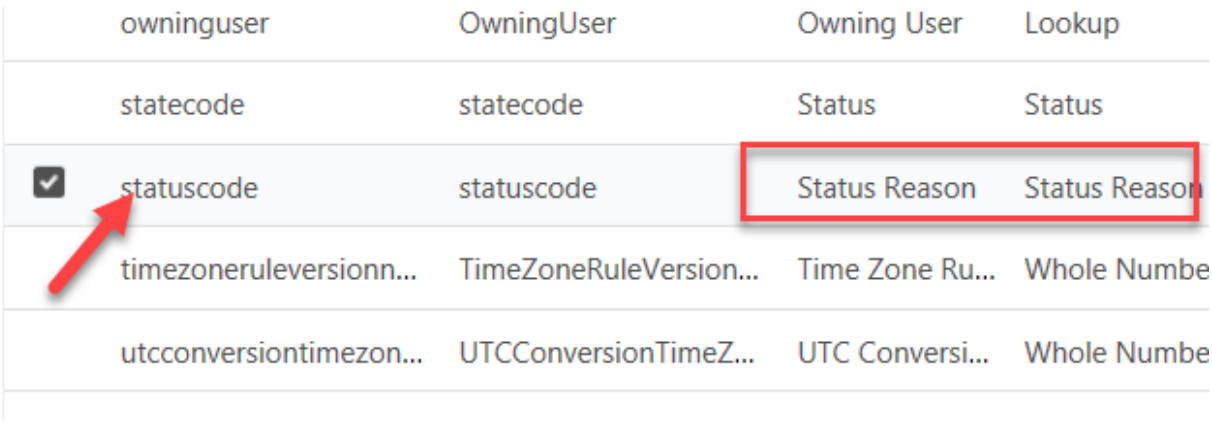
6.5 Task #5: Edit Status Reason Options

1. Open the Permit Management solution
   - Navigate to Power Apps maker portal
   - Select **Solutions** from the left menu and click to open the **Permit Management** solution.
2. Switch to Classic
   - Click on the **...** icon and select **Switch to Classic**.

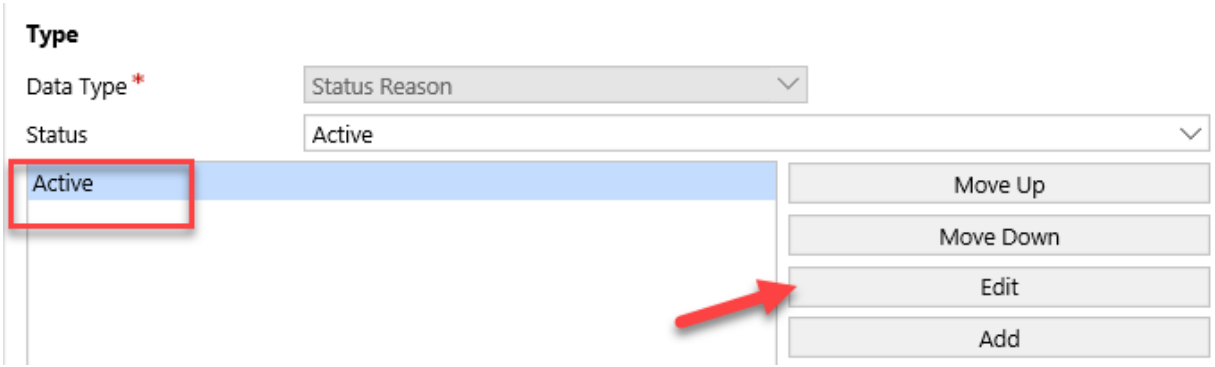


3. Edit Inspection Table Status Reason options
   - Expand **Entities**.
   - Expand the **Inspection** Table and select **Fields**.

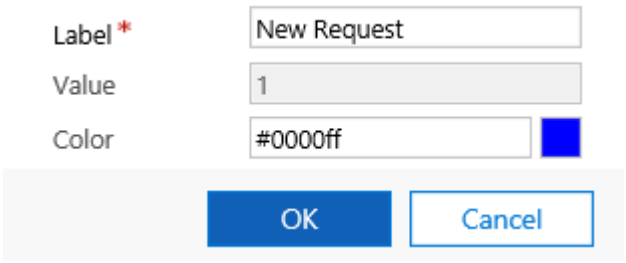
   - Locate and double click to open the **statuscode** Column.



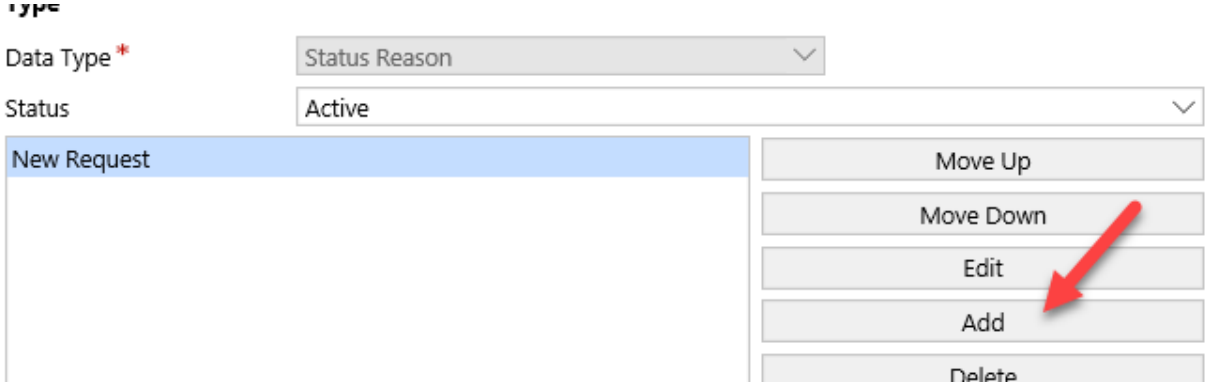
4. Change the Active option label
   - Make sure you have **Active** selected for **Status**.
   - Select the **Active** option and click **Edit**.



   - Change the **Label** to **New Request** and click **OK**.



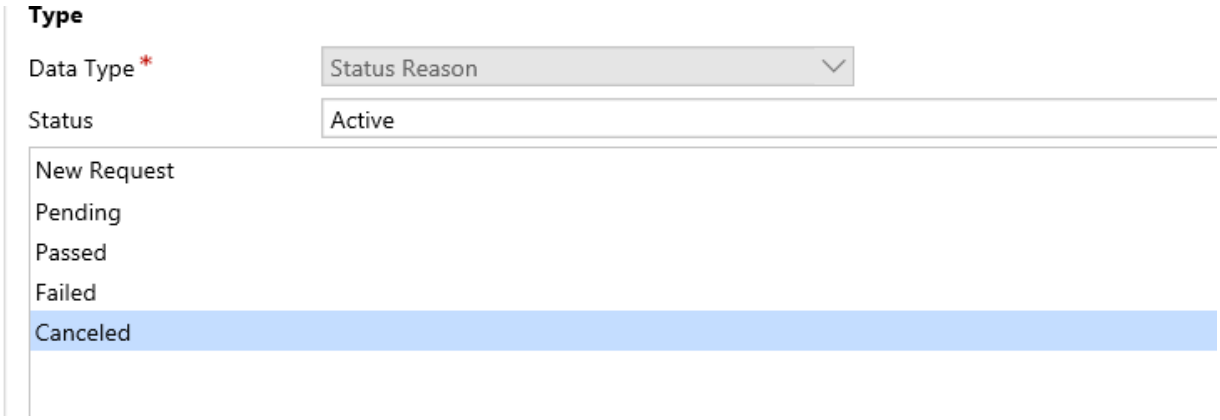
5. Add the Pending Option
   - Click **Add**.



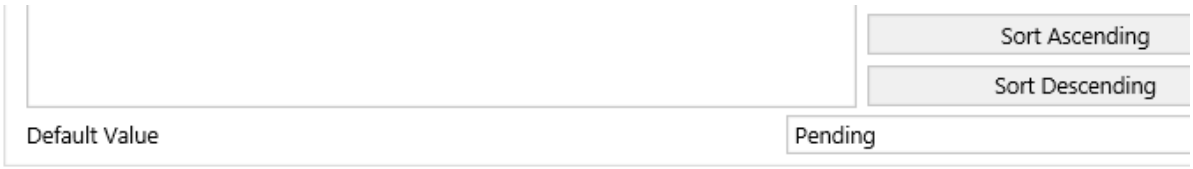
   - Enter **Pending** for **Label** and click **OK**.
6. Add the Passed Option
   - Click **Add**.

- Enter **Passed** for **Label** and click **OK.**

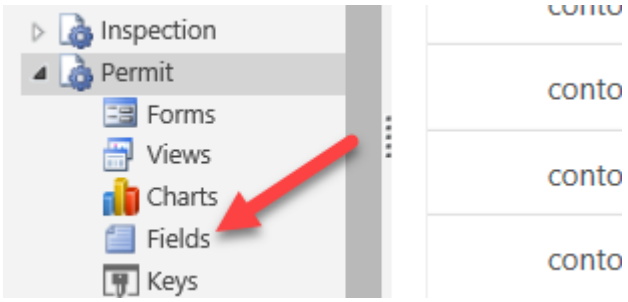
7. Add the Failed Option
   - Click **Add**.
   - Enter **Failed** for **Label** and click **OK.**
8. Add the Canceled Option
   - Click **Add**.
   - Enter **Canceled** for **Label** and click **OK.**
9. Your option-set should now have 5 options for the **Active** state.



10. Select Pending as the Default Value and click **Save and Close** from the top menu.



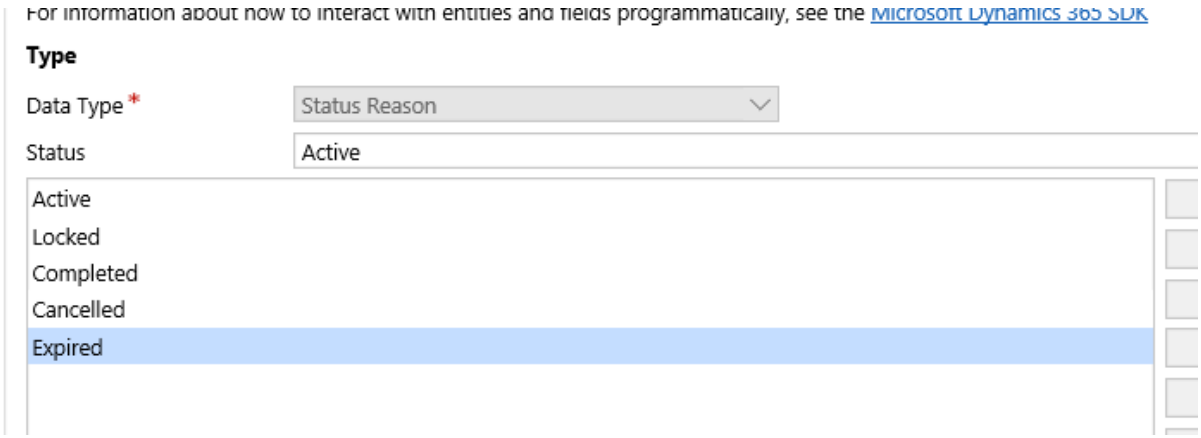
11. Edit Permit Table Status Reason options
    - Expand the **Permit** Table and select **Fields**.



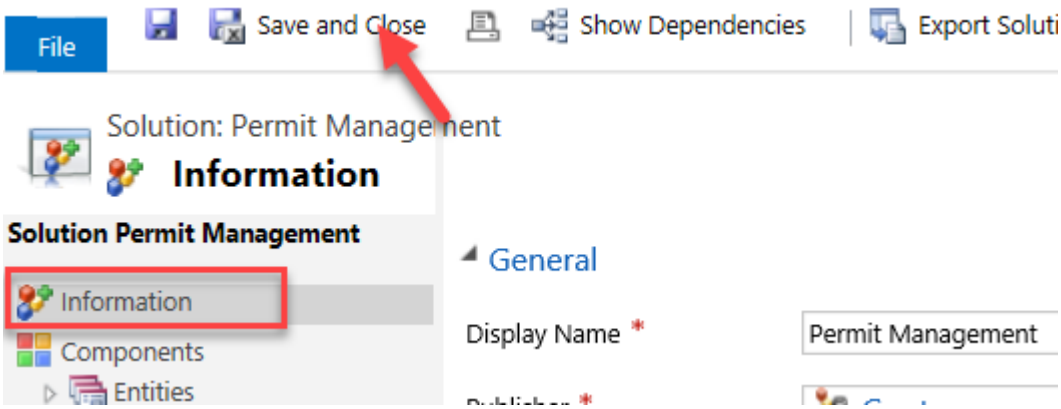
    - Locate and double click to open the **statuscode** Column.
12. Add the Locked option
    - Make sure you have the **Active** selected for **Status**.
    - Click **Add**.
    - Enter **Locked** for Label and click **OK**.
13. Add the Completed option
    - Click **Add**.
    - Enter **Completed** for Label and click **OK**.
14. Add the Canceled option
    - Click **Add**.

- Enter **Canceled** for Label and click **OK**.

15. Add the Expired option
    - Click **Add**.
    - Enter **Expired** for Label and click **OK**.
16. Your option-set should now have 5 options for the **Active** state



17. Select the **Active** for the **Default Value** and click **Save and Close** from the top menu
18. Select **Information** from the left side menu and click **Save and Close** to close classic solution explorer



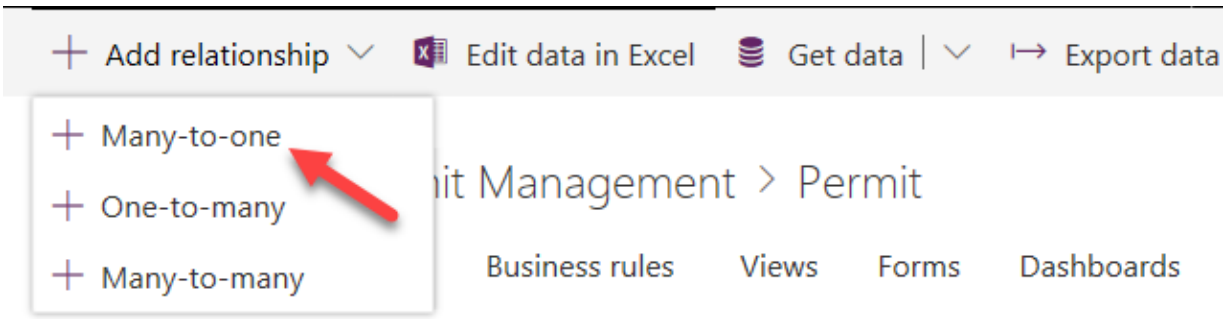
19. Select **Solutions** from the top menu and click **Publish All Customizations**.

7 Exercise #3: Create Relationships

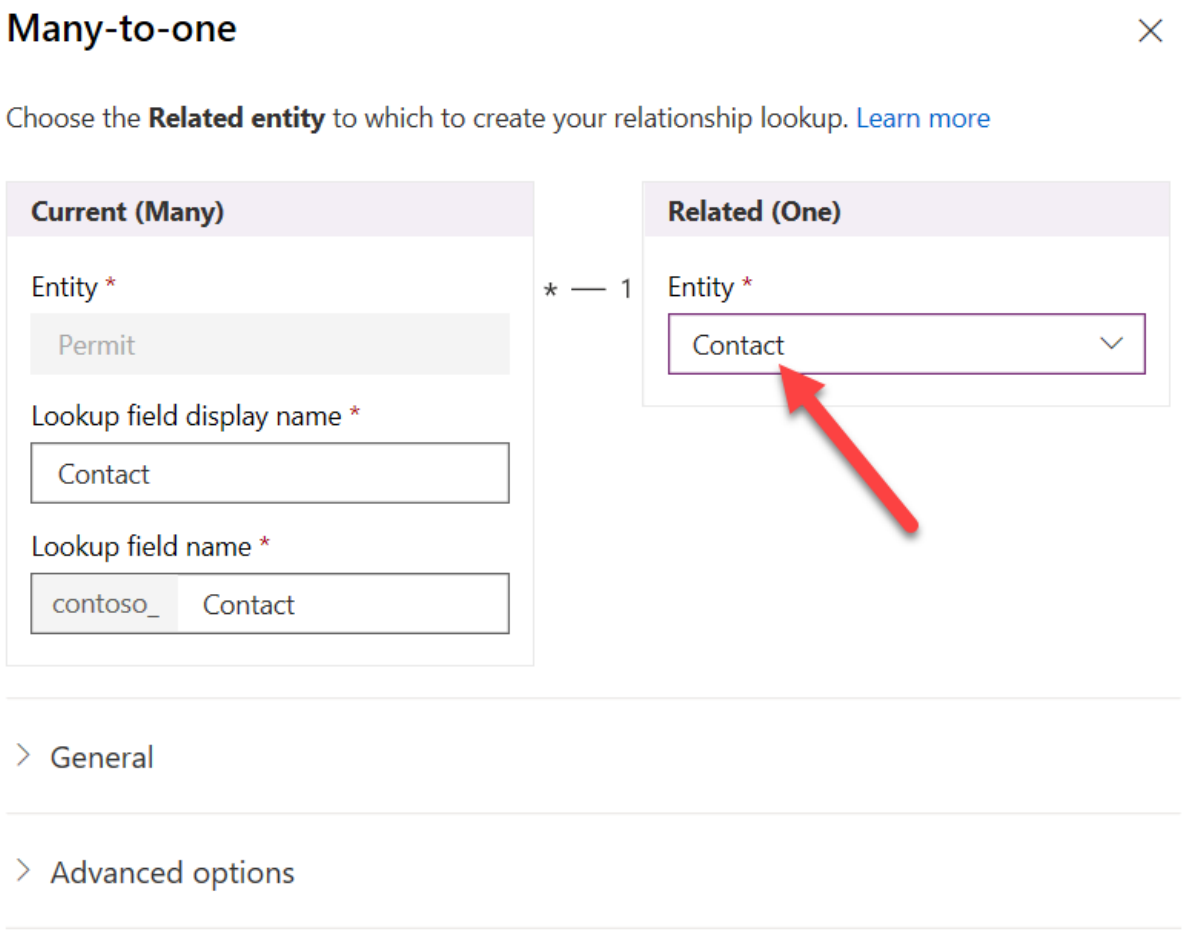
Objective: In this exercise, you will create relationships.

7.1 Task #1: Create Relationships

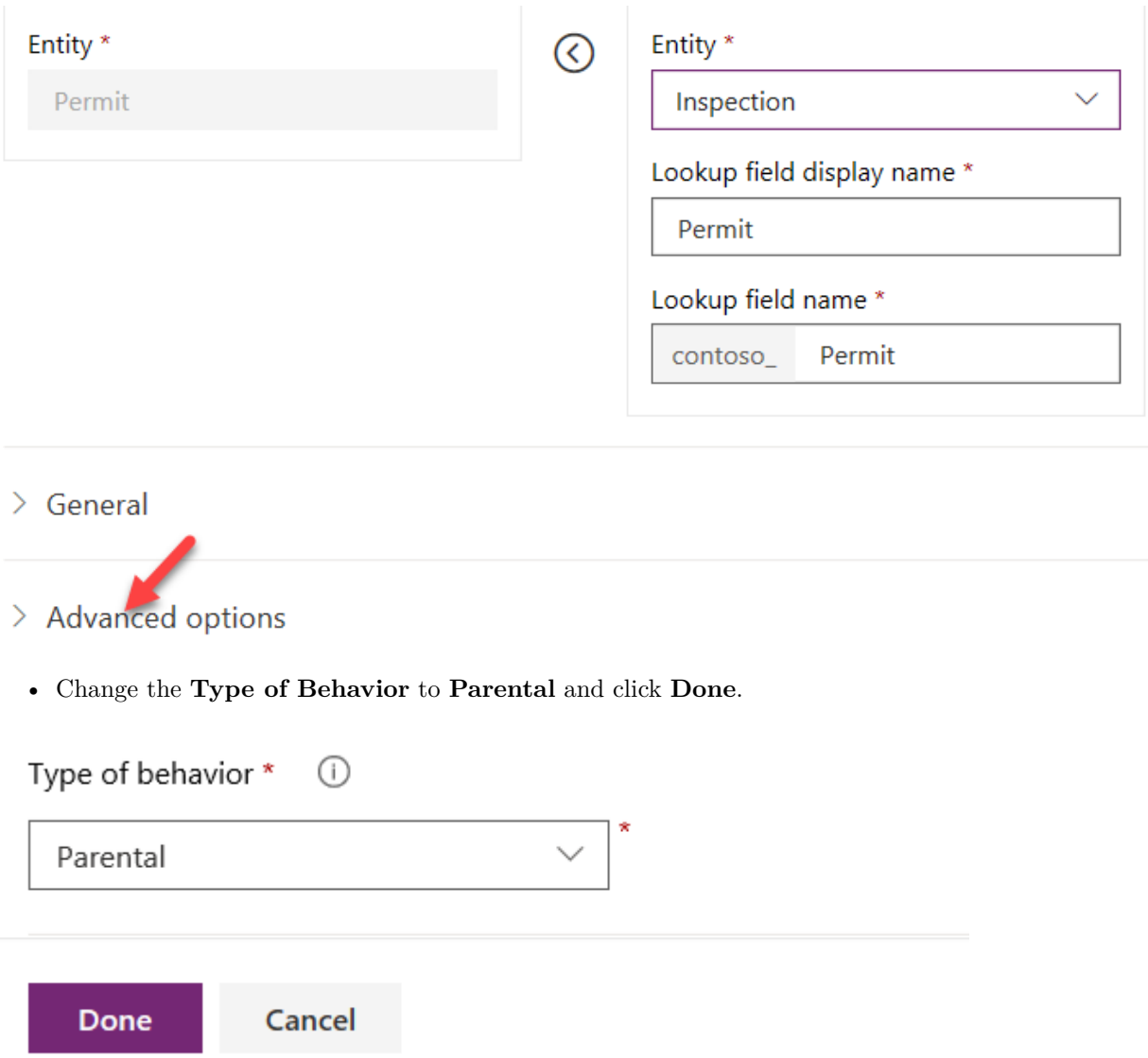
1. Open the Permit Management solution
    - Sign in to Power Apps maker portal
    - Select **Solutions** and click to open the **Permit Management** solution.
2. Create Permit to Contact relationship
    - Click to open the **Permit** Table.
    - Select the **Relationships** tab.
    - Click + **Add Relationship** and select **Many-to-one**.



- Select Contact for **Related (One)** and click **Done**.

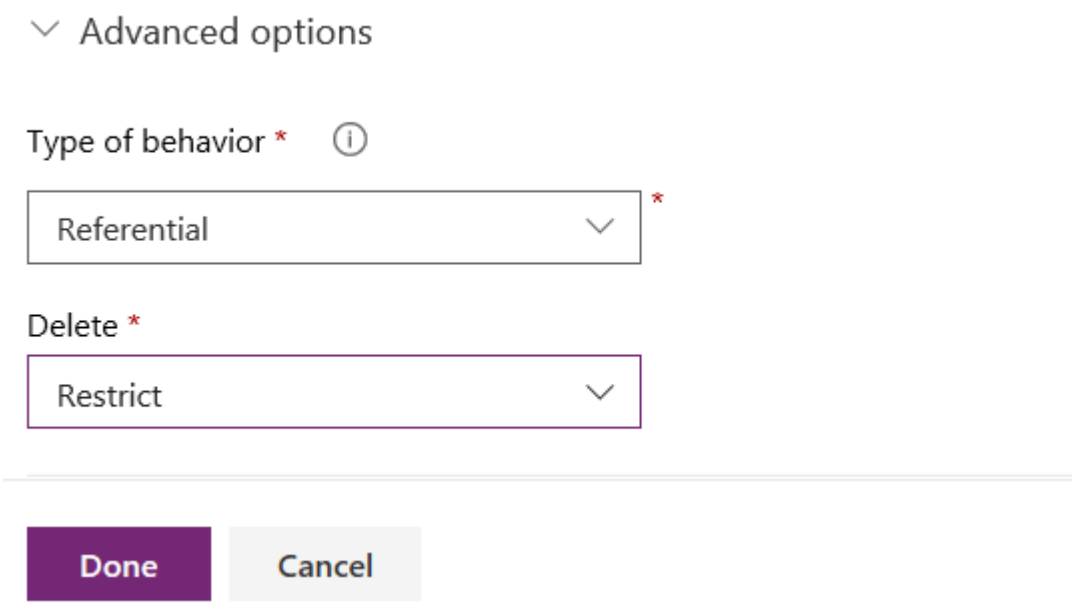


3. Create Permit to Inspection relationship
   - Click **Add Relationship** and select **One-to-Many**.
   - Select **Inspection** for **Table** in the **Related (Many)** and click **Advanced Options**.

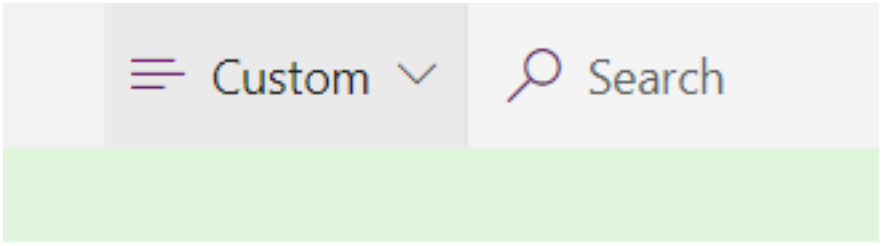


- Change the **Type of Behavior** to **Parental** and click **Done**.

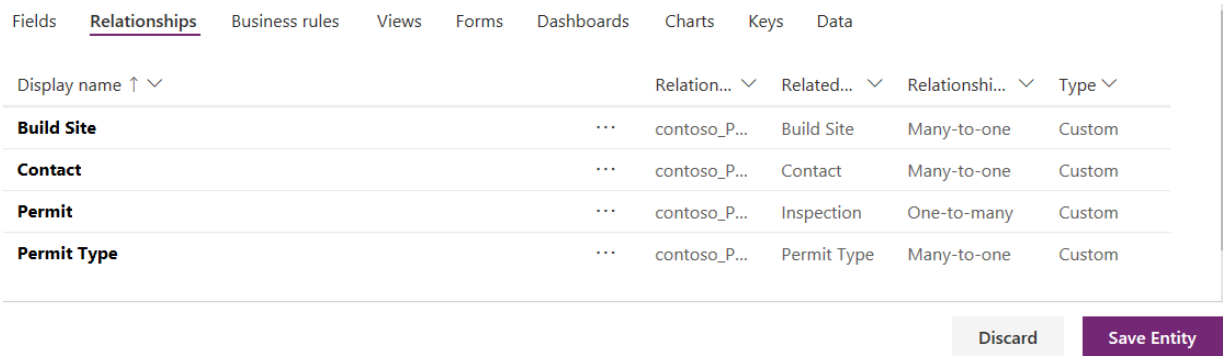
4. Create Permit to Build Site relationship
   - Click **Add Relationship** and select **Many-to-One**.
   - Select **Build Site** for **Related (One) Table** and click **Advanced Options**.
   - Change the **Delete** to **Restrict** and click **Done**.



5. Create Permit to Permit Type relationship
   - Click **Add Relationship** and select **Many-to-One**.
   - Select **Permit Type** for **Related (One) Table** and click **Done**.
6. Change the filter to **Custom**.



7. Click **Save Table**.



8. Select **Solutions** from the top menu and click **Publish All Customizations.**

8

8.1 lab: title: 'Lab 02: Model driven app'

[!NOTE] Effective November 2020:

- Common Data Service has been renamed to Microsoft Dataverse. Learn more

- Some terminology in Microsoft Dataverse has been updated. For example, *entity* is now *table* and *field* is now *column*. Learn more

This content will be updated soon to reflect the latest terminology.

8.2 Lab 02 – Model driven app

9 Scenario

A regional building department issues and tracks permits for new buildings and updates for remodeling of existing buildings. Throughout this course you will build applications and automation to enable the regional building department to manage the permitting process. This will be an end-to-end solution which will help you understand the overall process flow.

In this lab we will continue to build on top of the components created in the previous Lab. We will now build a Power Apps model-driven app to allow the office staff manage records for the inspectors and the inspectors to manage their own records as needed.

10 High-level lab steps

As part of creating the model-driven app, you will complete the following:

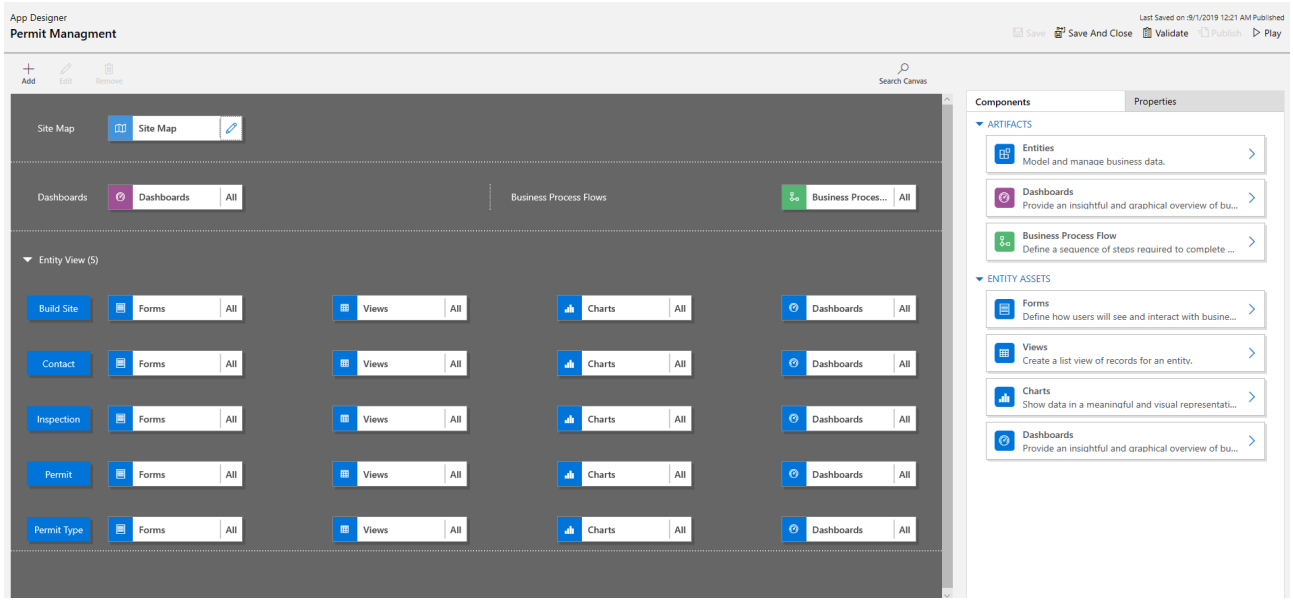
- Create a new model-driven app named Permit Management
- Edit the app navigation to reference the required Tables
- Customize the forms and views of the required Tables for the app

Views: As the name suggests, this helps viewing the existing data in the form of table. This is the configuration of the Columns that will be displayed on the screen.

Forms: This is where the user creates/updates new records in the Tables.

Both will be integrated to the model-driven app for a better user-experience.

The following is what the model-driven app designer looks like when all the customizations are completed:



10.1 Things to consider before you begin

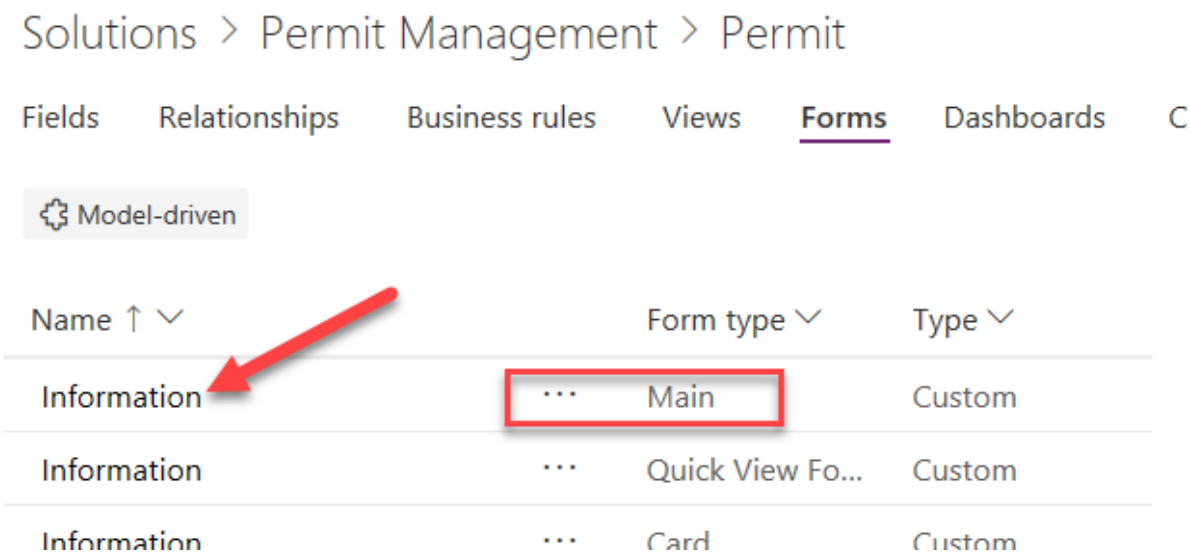
- What changes should we make to improve the user experience?
- What should we include in a model-driven app based on the data model we've built?
- What customizations can be made on the sitemap of a model-driven app?
- Remember to continue working in your DEVELOPMENT environment. We'll move everything to production once everything is built and tested.

11 Exercise #1: Customize Views and Forms

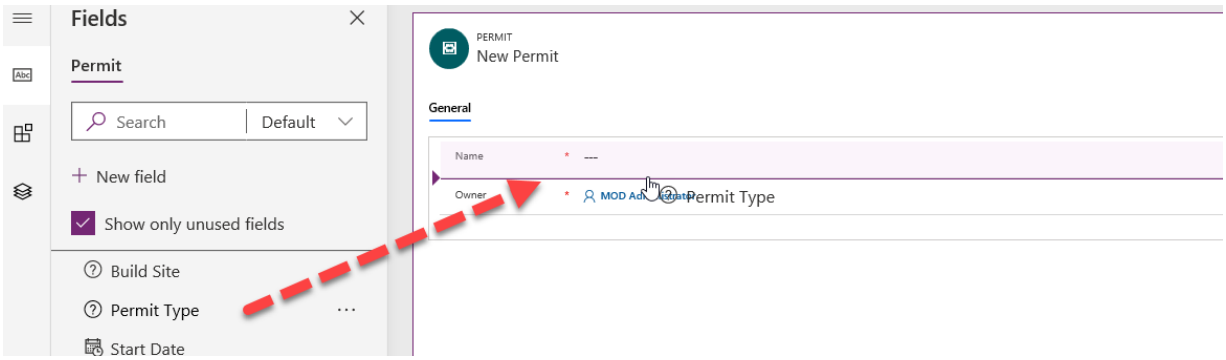
Objective: In this exercise, you will customize views and forms of the custom created Tables that will be used in the model-driven app.

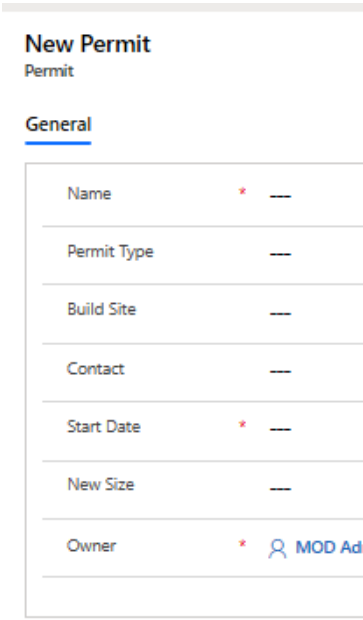
11.1 Task #1: Edit Permit Form and View

1. In your development environment, open the Permit Management solution.
   - Sign in to Power Apps maker portal
   - Select your **Dev environment.**
   - Select **Solutions**.
   - Click to open the **Permit Management** solution.
2. Steps to edit the Permit Table form.
   - Click to open the **Permit** Table.
   - Select the **Forms** tab and click to open the **Main** form. By default, the form has two Columns, Name (Primary Column) and Owner.

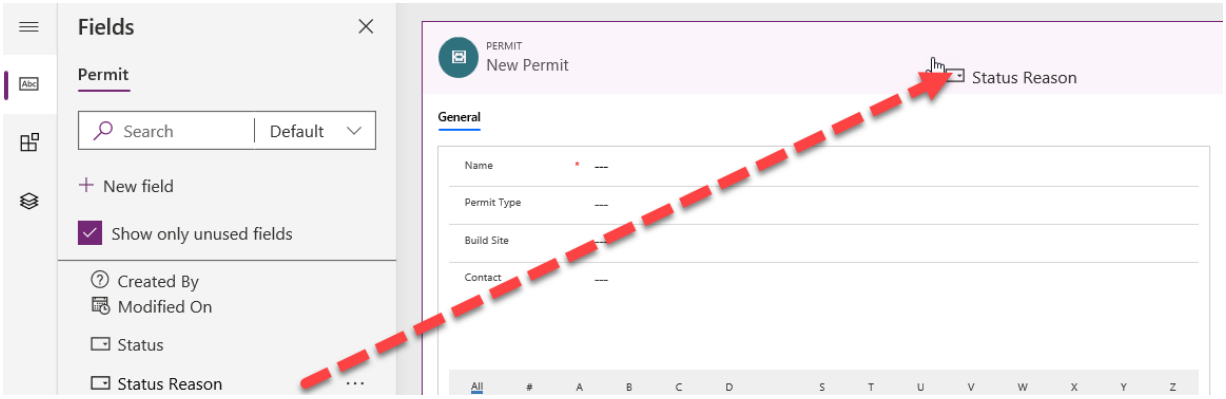


- Drag the **Permit Type** Column to the form and place it below the **Name** Column.

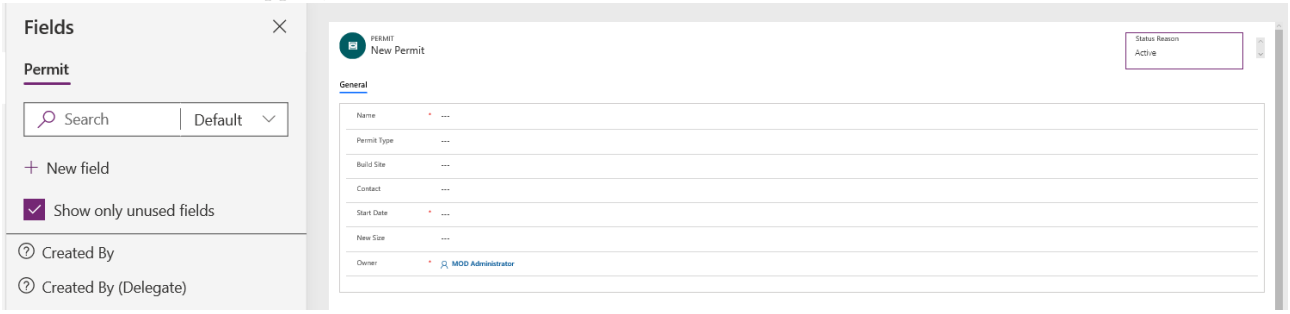




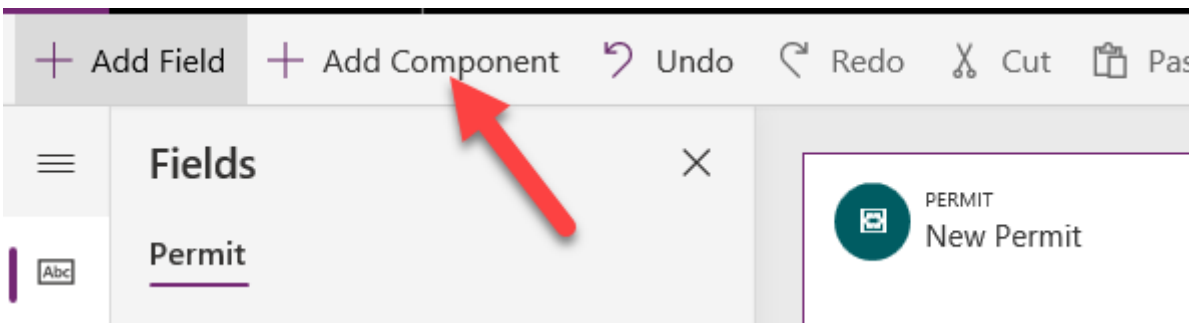
- Add **Build Site** lookup, **Contact** lookup, **Start Date** and **New Size** to the form.
- Drag the **Status Reason** Column and drop it in the right side of the form header.



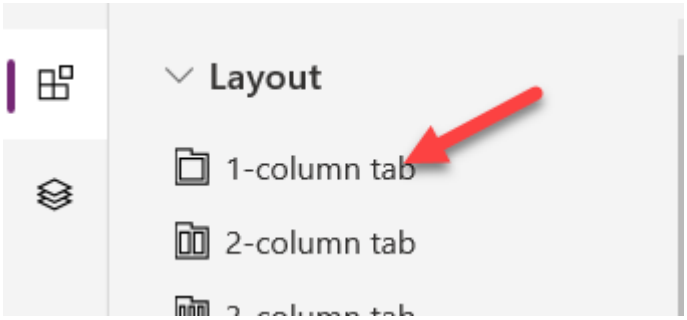
Once the control is dropped, this form will look like:



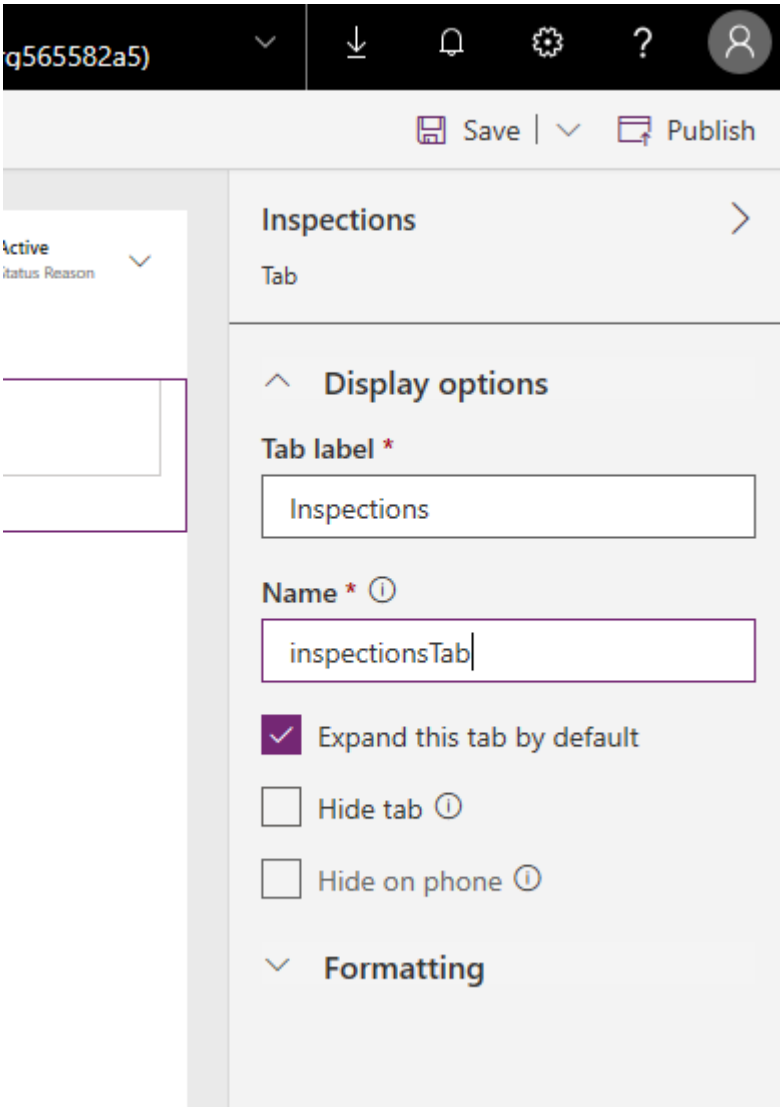
3. Add new tab for **Inspections** to the form.
   - With focus set on the main body of the form (not in the header) click **Add Component**.



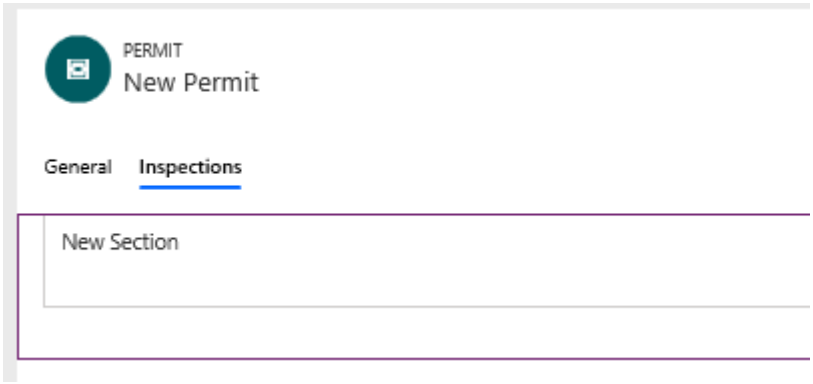
   - Select **One Column Tab**.



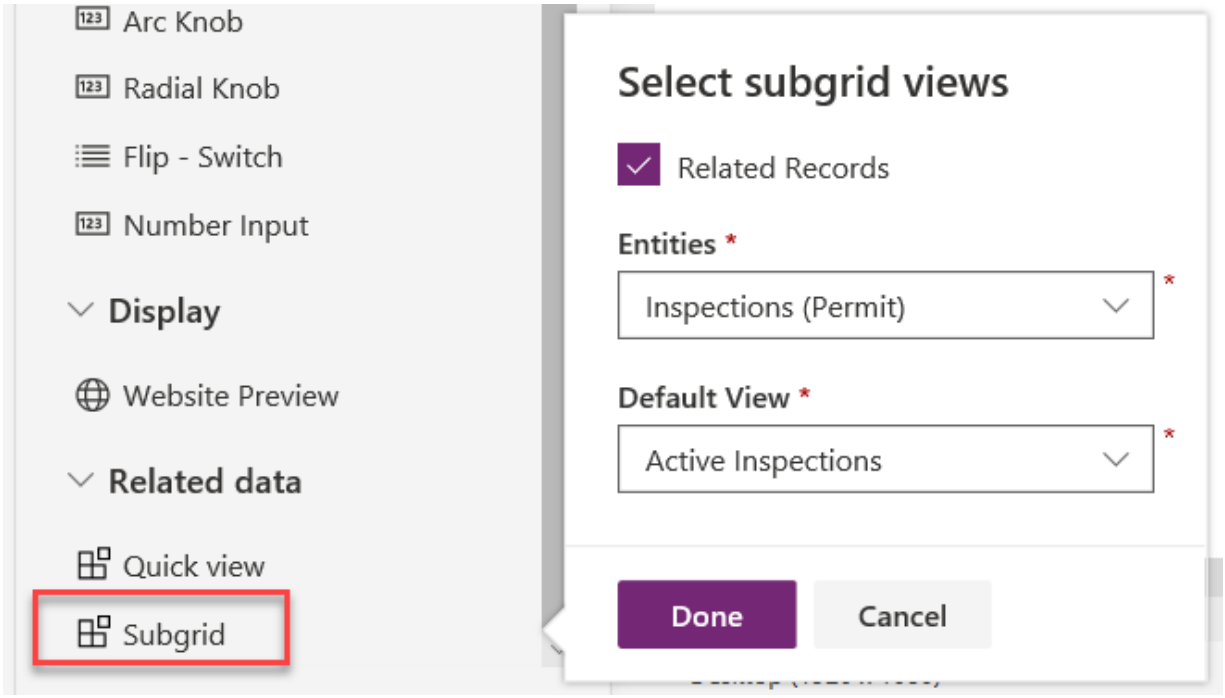
- Select the new tab you added.
- Go to the **Properties** pane, change the **Tab Label** to **Inspections** and the **Name** to **inspectionsTab**.



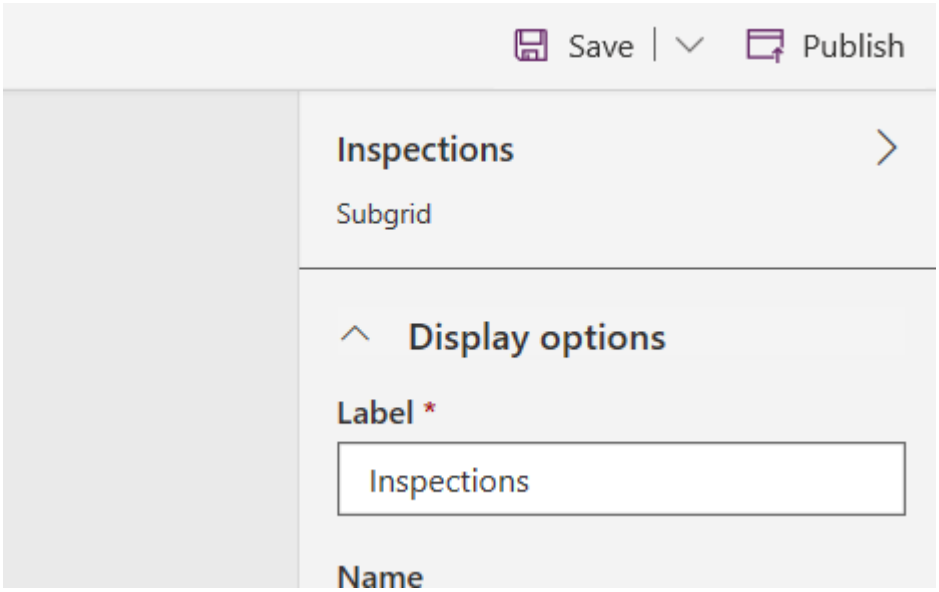
4. Steps to add Sub-Grid to the Permit form.
   - Select the **Inspections** tab. Make sure that you have selected the whole Tab and not just a section.



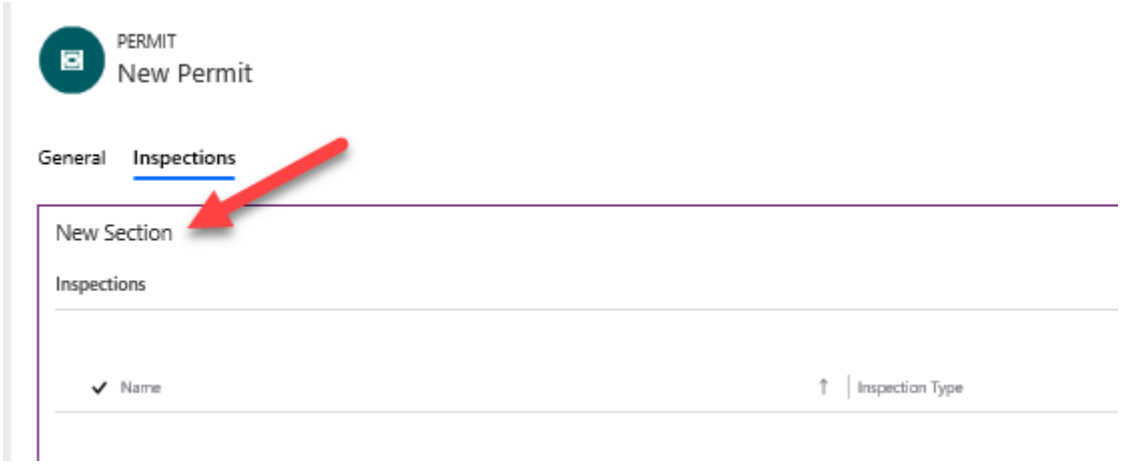
- Click **Add Component**.
- Scroll down and select **Subgrid,** this will open a pop-up to select Table.
- Check the **Show Related Records** checkbox, select **Inspections** for **Table**, select **Active Inspections** for **Default View** and click **Done**.



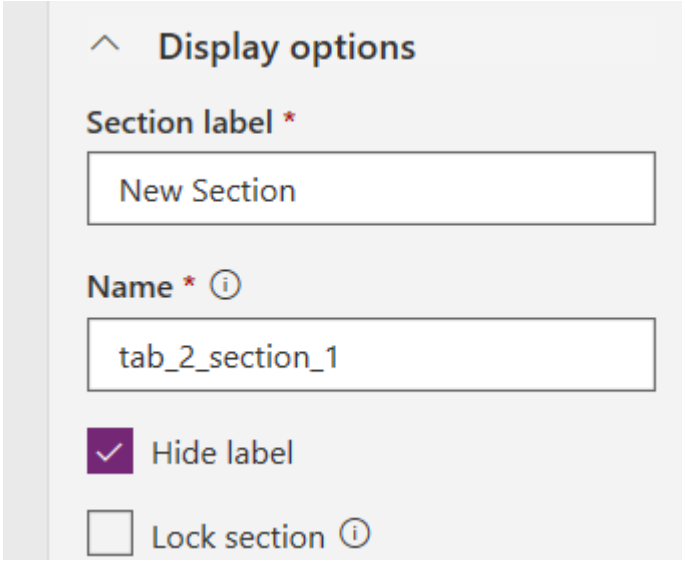
5. Edit Sub-Grid properties.
   - Go to the sub-grid properties pane and change the Label to **Inspections**.



6. Steps to hide the section label
   - Click to select the section.

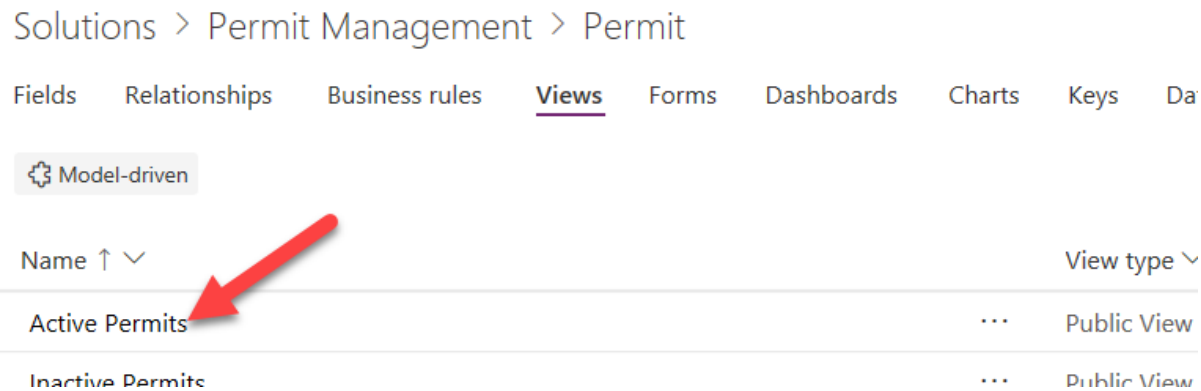


   - Go to the **Properties** pane and check the **Hide Label** checkbox.

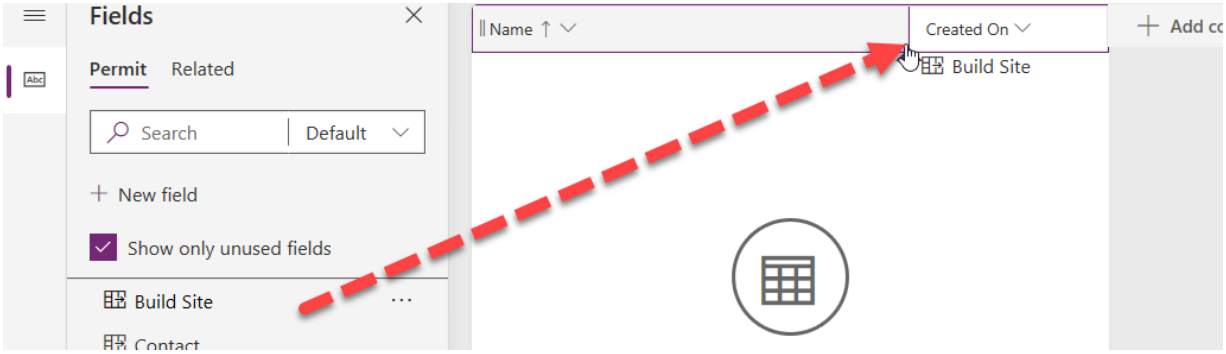


7. Click **Save** and wait for the save to complete.
8. Click **Publish** and wait for the publishing to complete.
9. Click on the **<- Back** button. You should now be back to the Permit Table Forms tab.

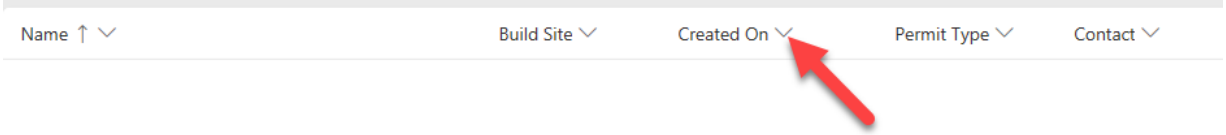
10. Steps to edit the Active Permits view.
    - Select the **Views** tab and click to open the **Active Permits** view.



    - Drag the **Build Site** Column and drop it between the **Name** and **Created On** Columns.



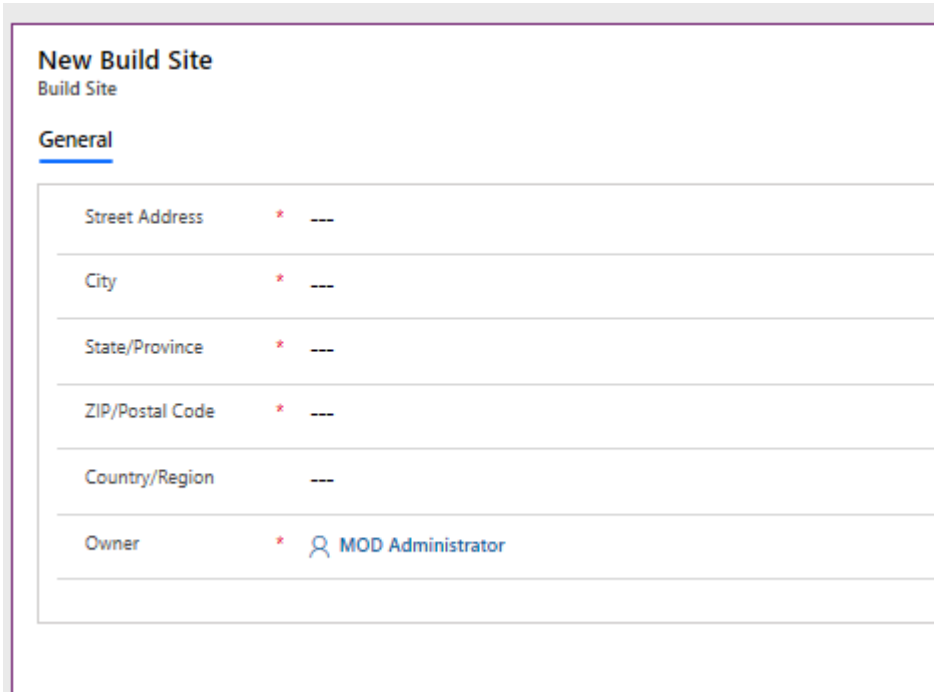
    - Click on the **Permit Type** Column. The Permit Type Column will be added to the view.
    - Click on the **Contact** Column. The **Contact** Column will be added to the view.
    - Go to the view designer and click on the chevron icon of the **Created On** column.



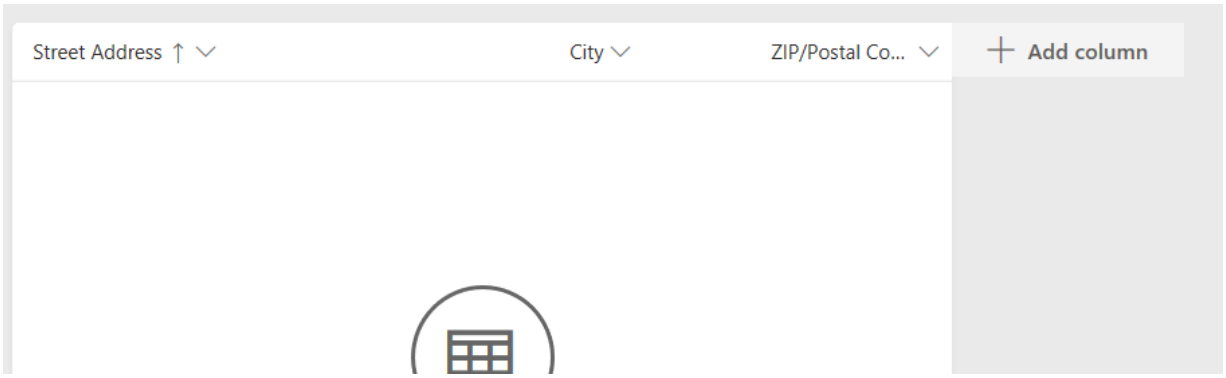
    - Click **Remove**. **Created On** Column will now be removed from the view.
    - Click **Save** and wait until the changes are saved.
    - Click **Publish** and wait for the publishing to complete.
11. Click on the **<-Back** button.

11.2 Task #2: Edit Build Site Form and View

1. Open the Permit Management solution.
    - Sign in to Power Apps maker portal
    - While in your dev environment, select **Solutions**, and click to open the **Permit Management** solution.
2. Edit the Build Site Table form.
    - Click to open the **Build Site** Table.
    - Select the **Forms** tab and click to open the **Main** form.
    - Add **City**, **State/Province**, **Zip/Postal Code**, and **Country Region** Columns to the form between **Street Address** and **Owner**.



3. Click **Save** and wait until the changes are saved.
4. Click **Publish** and wait for the publishing to complete.
5. Click on the **<-Back** button.
6. Edit the Active Build Sites view.
   - Select the **Views** tab and click to open the **Active Build Sites** view.
   - Add **City** and **Zip/Postal Code** to the view.
   - Remove **Created On** from the view by selecting **Remove** from the options in Column Chevron.

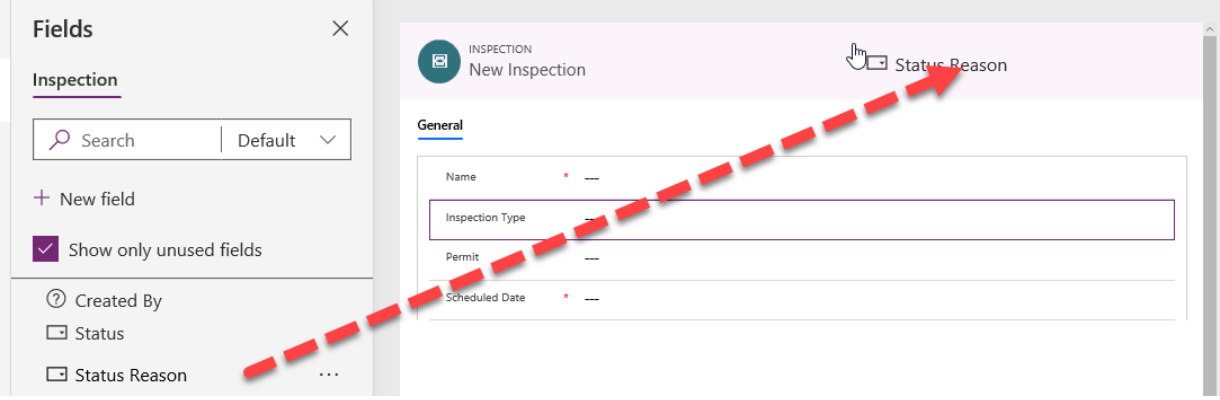


7. Click **Save** and wait until the changes are saved.
8. Click **Publish** and wait for the publishing to complete.
9. Click on the **<-Back** button.

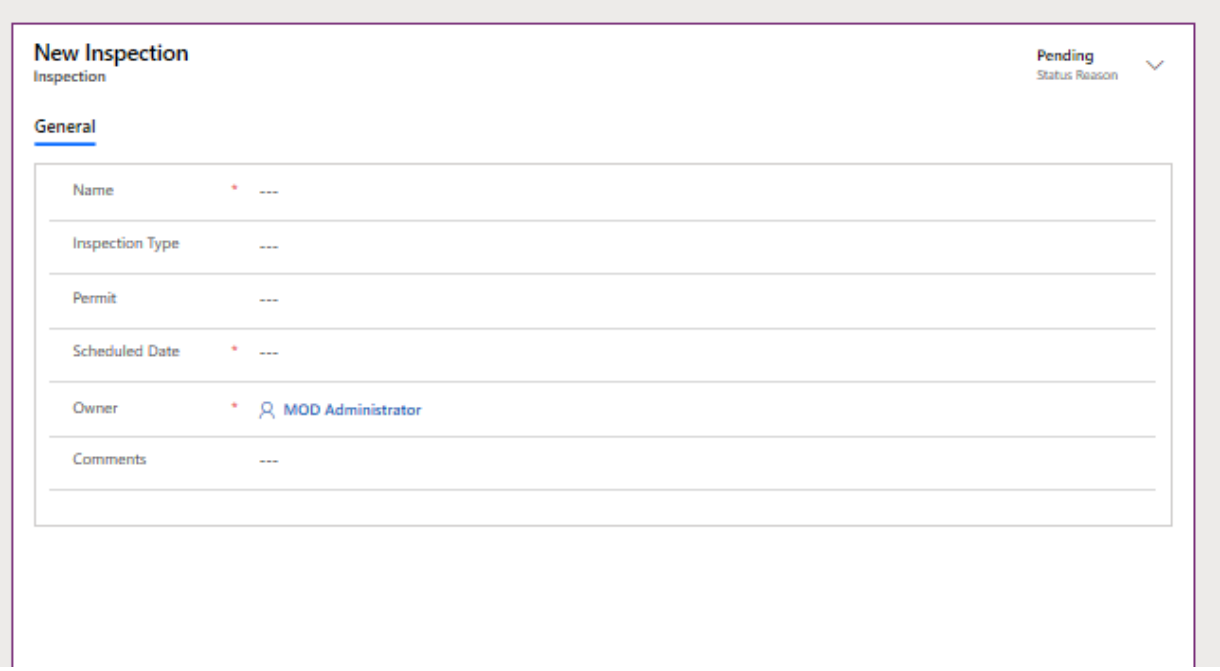
11.3 Task #3: Edit Inspection Form and View

1. Open the Permit Management solution.
   - Sign in to Power Apps maker portal
   - While in your dev environment, select **Solutions** and click to open the **Permit Management** solution.
2. Edit the Inspection Table form.
   - Click to open the **Inspection** Table.

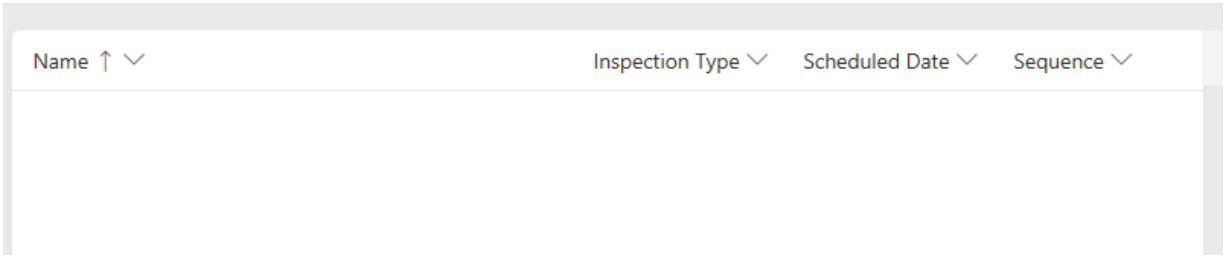
- Select the **Forms** tab and click to open the **Main** form.
- Add **Inspection type**, **Permit**, **Scheduled Date**, and **Comments** Columns to the form. **Inspection type**, **Permit**, **Scheduled Date** should be added between **Name** and **Owner**, while **Comments** will be added after the **Owner** Column.
- Drag the **Status Reason** Column **Comments** and drop it in the right side of the form header.



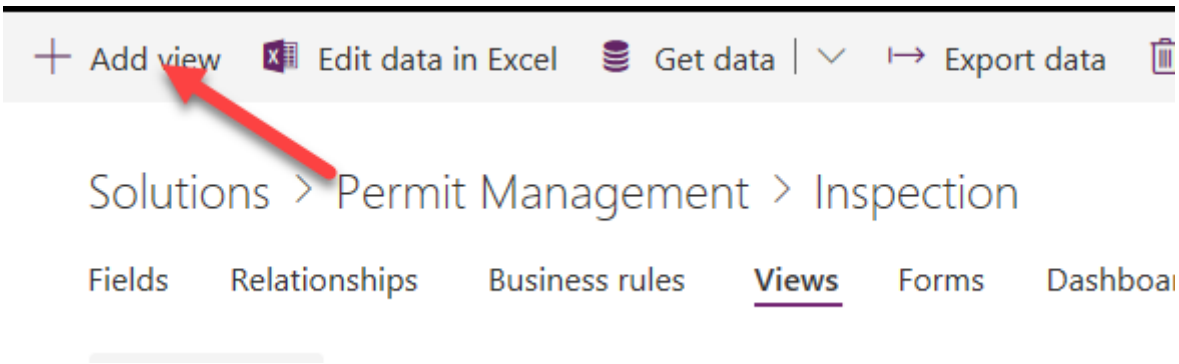
- The form should now look like the image below.



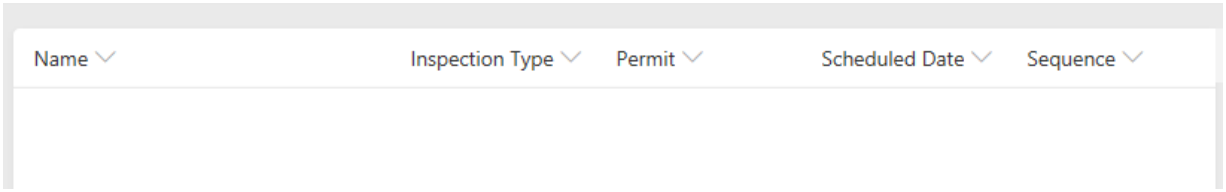
3. Click **Save** and wait until the changes are saved.
4. Click **Publish** and wait for the publishing to complete.
5. Click on the **<-Back** button.
6. Edit the Active Inspections view.
   - Select the **Views** tab and click to open the **Active Inspections Sites** view.
   - Add **Inspection Type, Scheduled Date** and **Sequence** to the view.
   - Remove **Created On** from the view by selecting the chevron on the Column and select **Remove**.



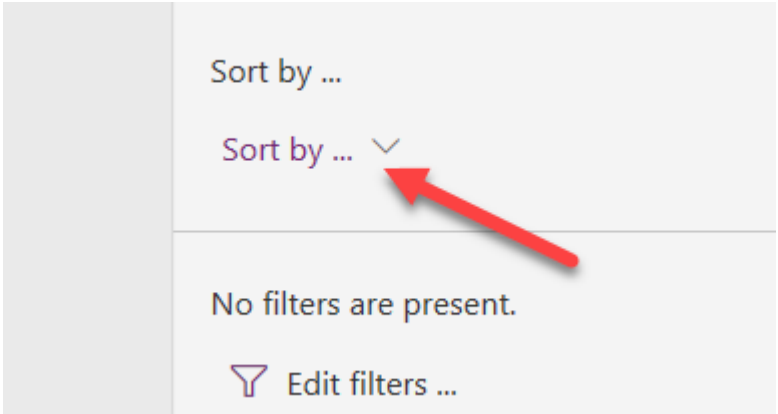
7. Click **Save** and wait until the changes are saved.
8. Click **Publish** and wait for the publishing to complete.
9. Click on the **<-Back** button.
10. Create new Inspector View for the Inspection Table.
    - Make sure you still have the **Views** tab selected.
    - Click + **Add View**. This will open a new window to create View.



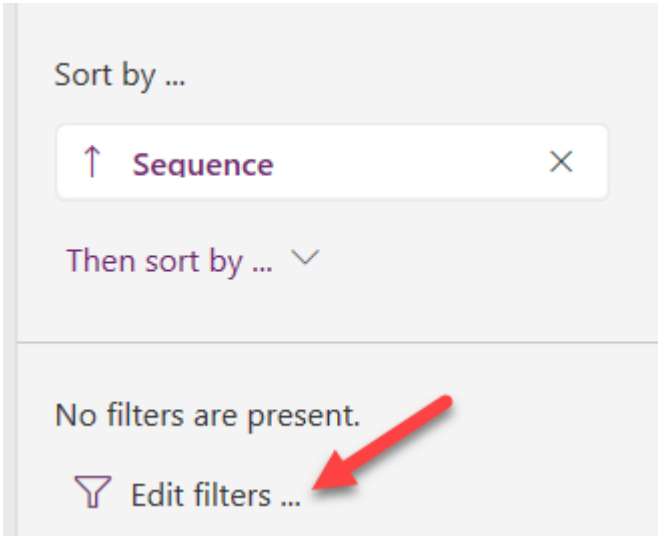
    - Enter **Inspector View** for **Name** and click **Create**.
    - Add **Inspection Type**, **Permit**, **Scheduled Date**, and **Sequence** Columns to the view.



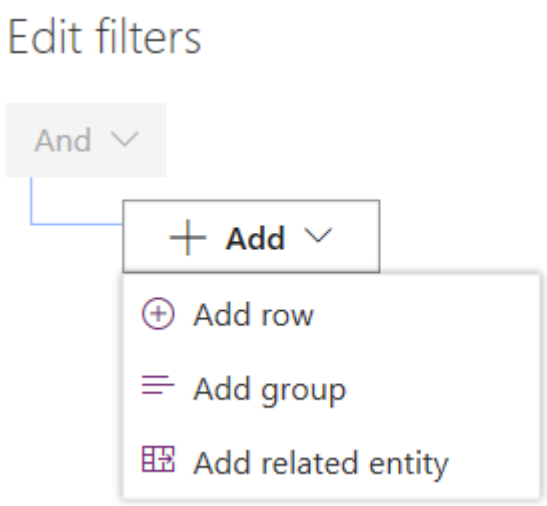
11. Sort the Inspector View by the sequence.
    - Go to the view properties pane and click **Sort By**.



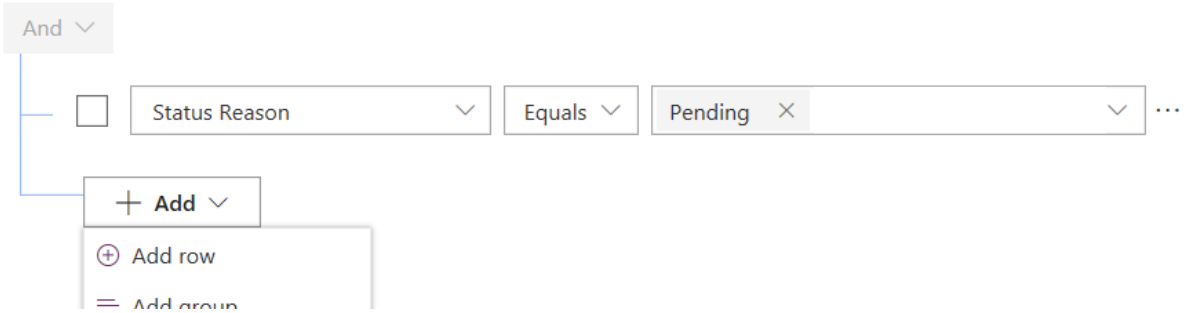
    - Select **Sequence**.
12. Filter the Inspector View.
    - Go to the view properties pane and click **Edit Filter**. This will open a new pop-up on the right side of the window.



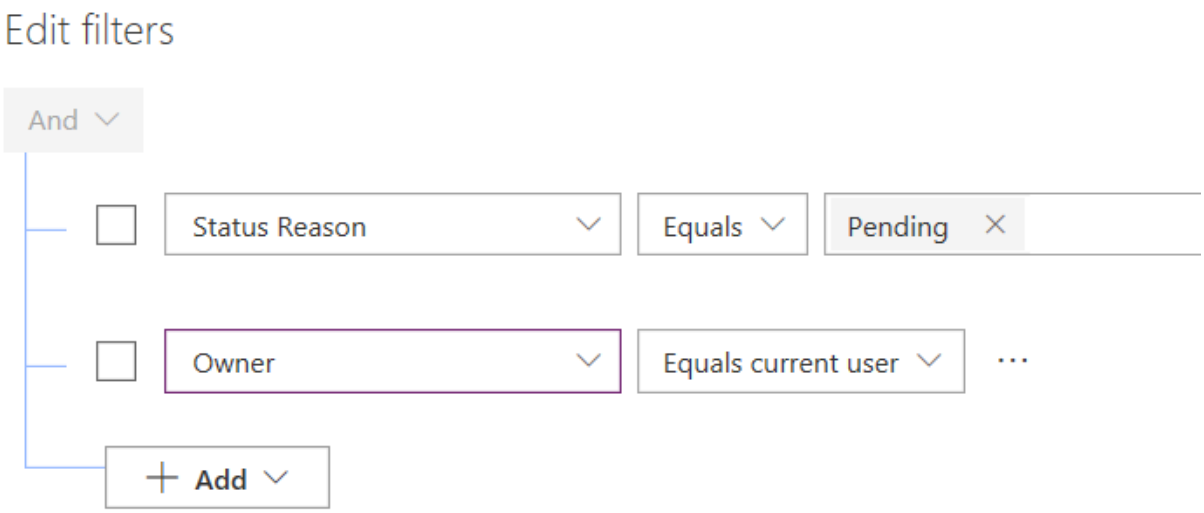
- Click **Add** and select **Add Row**.



- Set the filter property by Selecting **Status Reason** in first dropdown and **Pending** in the third dropdown**.** Now, click **Add** and select **Add Row** again.



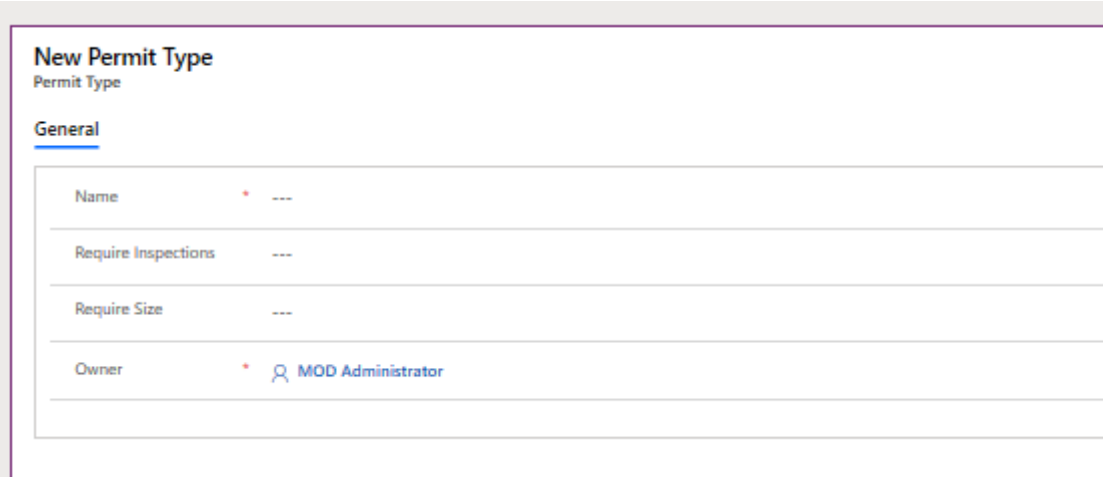
- To set the filter property select **Owner** Column in the first dropdown and Equals current user in second dropdown and click **OK**.



13. Click **Save** and wait until the changes are saved.
14. Click **Publish** and wait for the publishing to complete.
15. Once the changes are published, close the window and click **Done** on the previous window for the Power Apps.

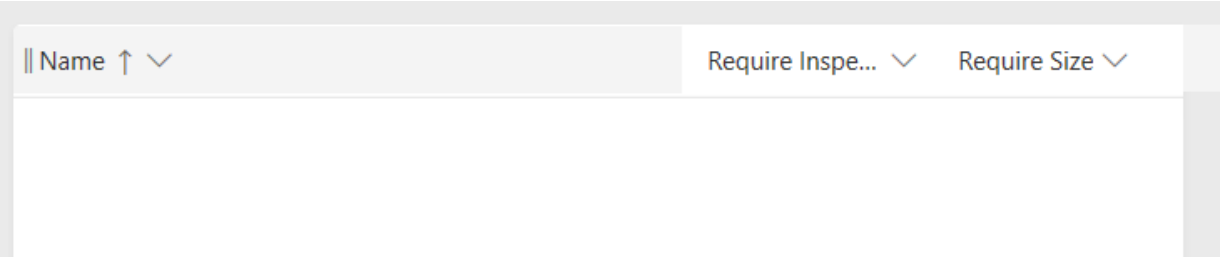
11.4 Task #4: Edit Permit Type Form

1. Open the Permit Management solution.
   - Sign in to Power Apps maker portal
   - Select your **Dev environment.**
   - Select **Solutions**.
   - Click to open the **Permit Management** solution.
2. Edit the Permit Type Table form.
   - Click to open the **Permit Type** Table.
   - Select the **Forms** tab and click to open the **Main** form.
   - Add **Require Inspections** and **Require Size** Columns to the form between **Name** and **Owner**.



   - Click **Save** and wait until the changes are saved.
3. Click **Publish** and wait for the publishing to complete.
   - Click on the **<-Back** button.
4. Edit the **Permit Type** Table **Active Permit Type** View.

- Select the **Views** tab and click to open the **Active Permit Type** view.
- Add **Require Inspections** and **Require Size** to the view.
- Remove **Created On** from the view but selecting the chevron on the Column and select **Remove**.



- Click **Save** and wait until the changes are saved.
- Click **Publish** and wait for the publishing to complete.
- Click on the **<-Back** button.

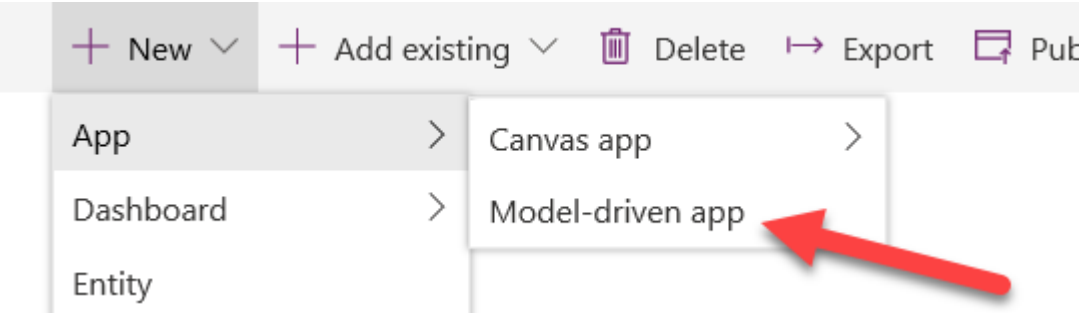
12 Exercise #2: Create Model-Driven Application

Objective: In this exercise, you will create the model-driven app, customize the sitemap, and test the app.

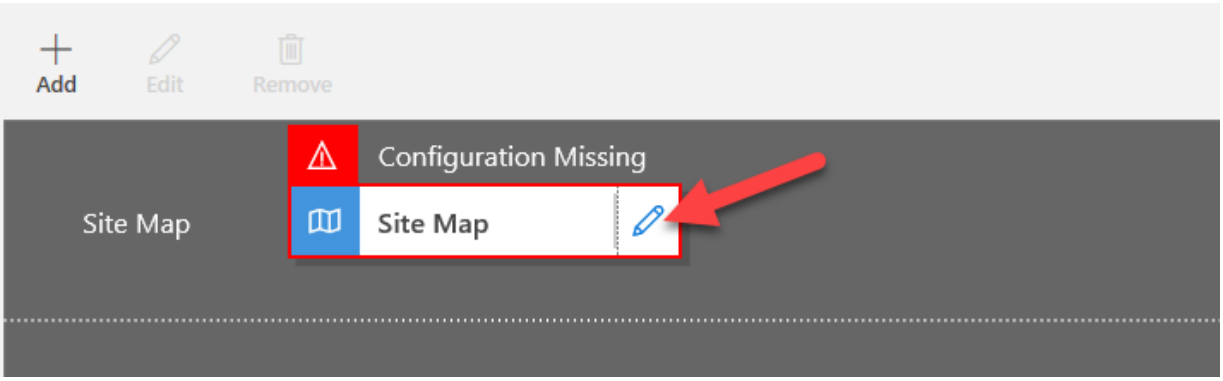
Note: You will see several Columns not addressed as you build out your application, particularly on the sitemap steps. We have taken some short cuts in the interest of time for doing the labs. In a real project you would give these items logical names.

12.1 Task #1: Create Application

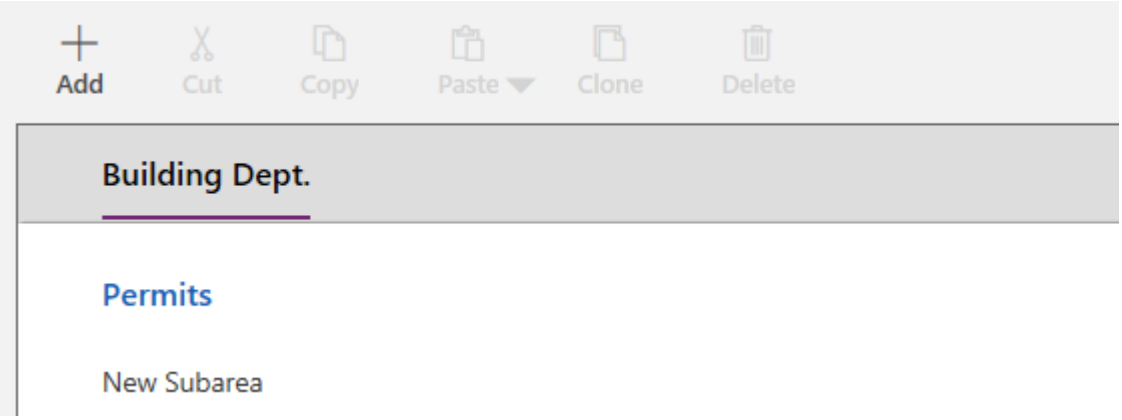
1. Open the Permit Management solution.
   - Sign in to Power Apps maker portal
   - While in your dev environment, click to open the **Permit Management** solution.
2. Create the Model-Driven Application
   - Click **New** and select **App | Model-Driven App**. This will open a new Window.



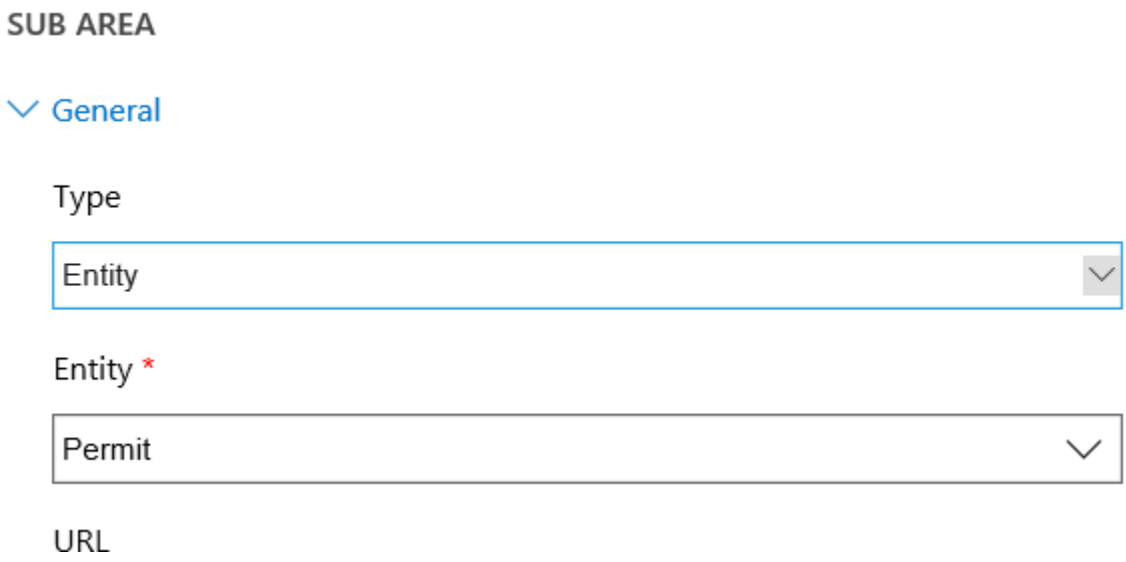
   - Enter **Permit Management** for **Name** and click **Done**.
3. Edit Sitemap
   - Click **Edit Site map.**



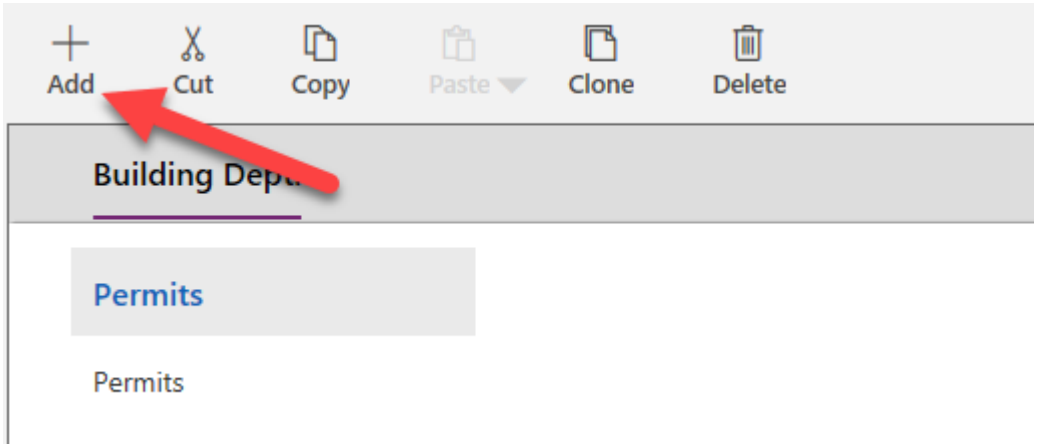
4. Edit the default titles
   - Select **New Area**.
   - Go to the properties pane and enter **Building Dept.** for **Title**.
   - Select **New Group**.
   - Go to the **Properties** pane and enter **Permits** for **Title**.



5. Add the Permit Table to the sitemap
   - Select **New Subarea**.
   - Go to the **Properties** pane and select **Table** from the dropdown for **Type**.
   - Search for **Permit** Table from the dropdown for **Table**.

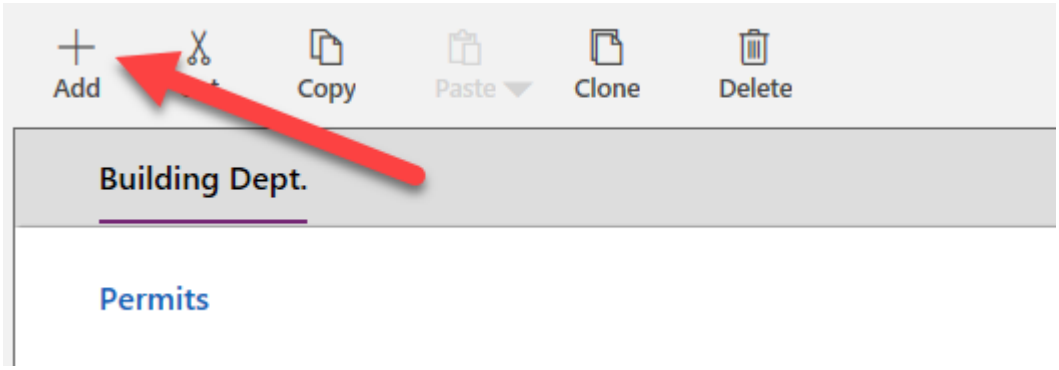


6. Add the Inspection Table to the sitemap
   - Select **Permits** group and click **Add**.



- Select **Subarea**.
- Go to the **Properties** pane.
- Select **Table** from the dropdown for **Type** and search for **Inspection** Table from the dropdown for **Table**.

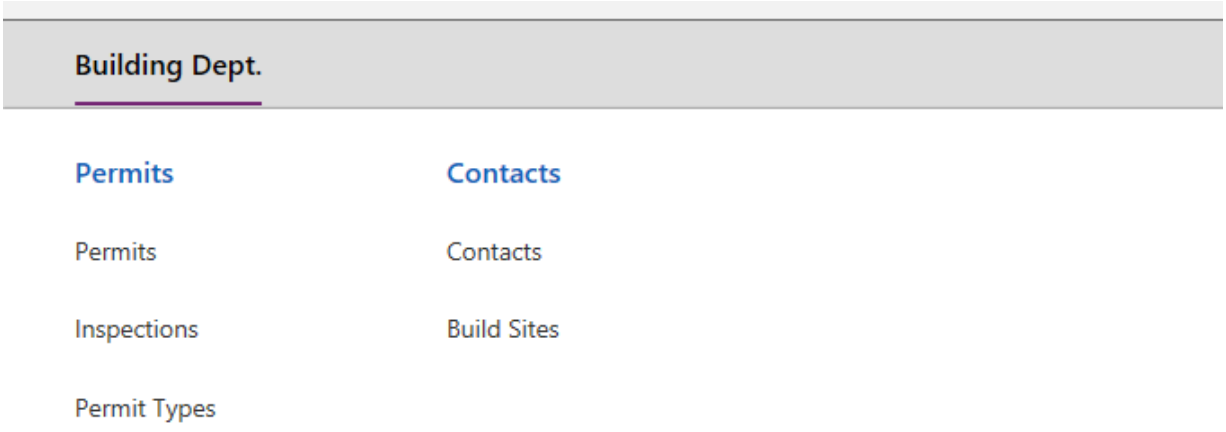
7. Add the Permit Type Table to the sitemap
   - Select **Permits** group and click **Add**.
   - Select **Subarea**.
   - Go to the **Properties** pane.
   - Select **Table** from the dropdown for **Type** and search for **Permit Type** Table from the dropdown for **Table**.
8. Add new Group to the sitemap
   - Select the **Building Dept.** area and click **Add.**



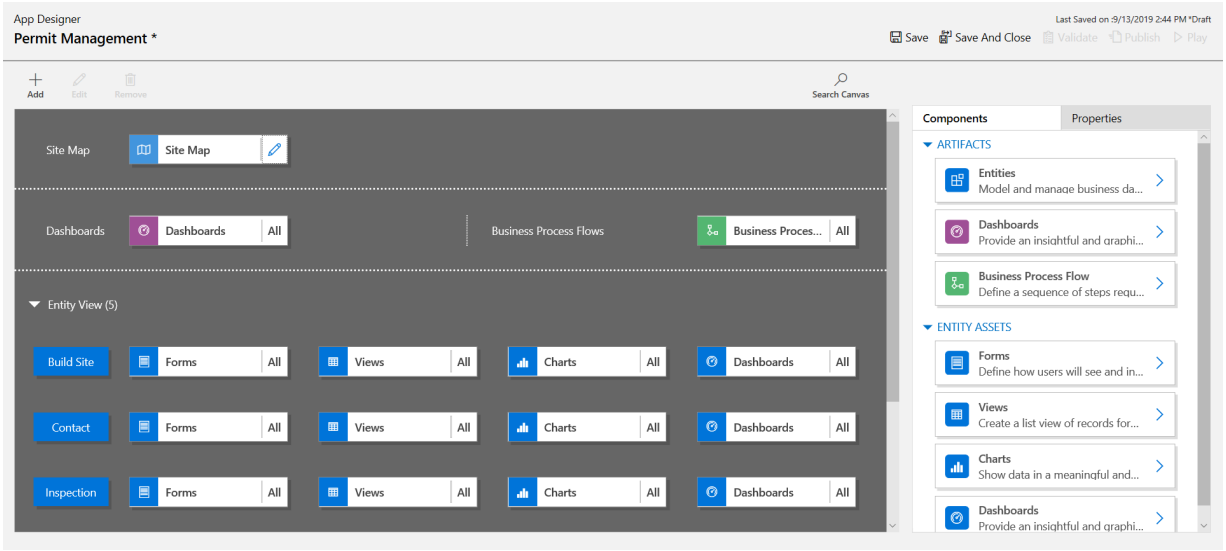
   - Click **Add** and select **Group**.
   - Select **Group**.
   - Go to the **Properties** pane and enter **Contacts** for Title.
9. Add the Contact Table to the Contacts group.
   - Select the **Contacts** group.
   - Click **Add** and select **Subarea.**
   - Go to the **Properties** pane.
   - Select **Table** from the dropdown for **Type** and search for **Contact** Table in the dropdown for **Table**.
10. Add the Build Site Table to the Contacts group.
    - Select the **Contacts** group.
    - Click **Add** and select **Subarea.**
    - Go to the **Properties** pane.

- Select **Table** from the dropdown for **Type** and search for **Build Site** Table in the dropdown for **Table**.

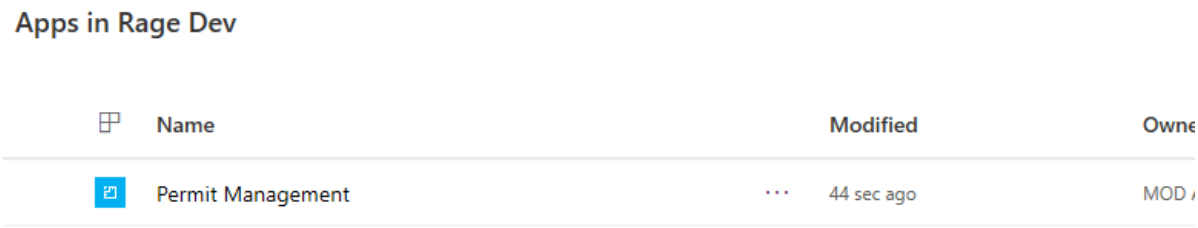
11. The sitemap should now look like the image below.



12. Click **Save**, this will show the loading screen while the changes are getting saved.
13. Click **Publish** to publish the sitemap and wait for the publishing to complete.
14. Click **Save and Close** to close the sitemap editor.
15. You will see the assets for the Tables that were added to the sitemap are now all in the application.

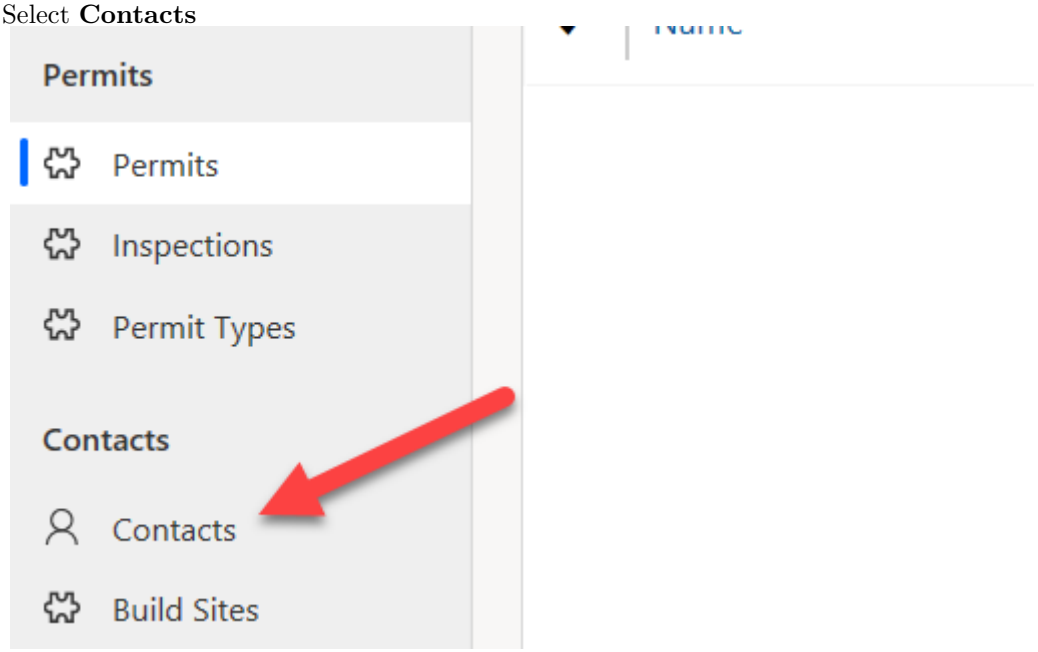


16. Click **Save** to save the application.
17. Click **Validate** to validate the changes done in the application. This will show some warnings. Feel free to review them, but we can ignore them, since we have not referenced a specific View and Form for the Tables and this will display all the Views and Forms.
18. Click **Publish** to publish the application and wait for the publishing to complete.
19. Click **Save and Close to** close the app designer.
20. Click **Done**.
21. Click **Publish all Customizations.**
22. Select **Apps** and your application should now be listed in the Recent Apps.

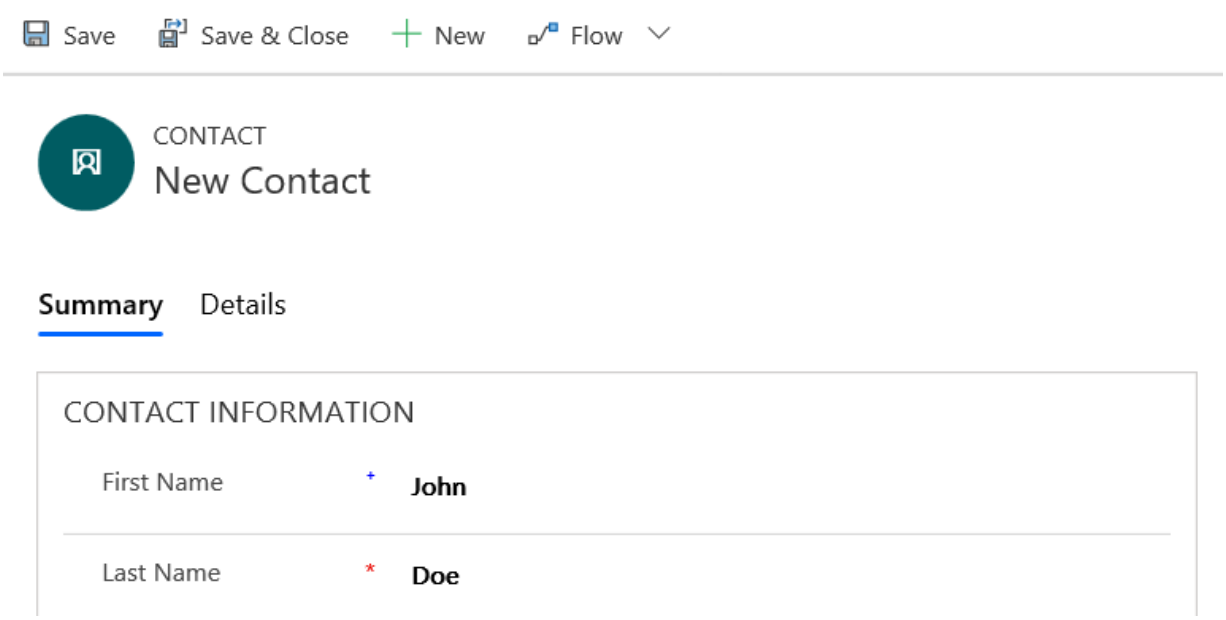


12.2 Task #2: Test Application

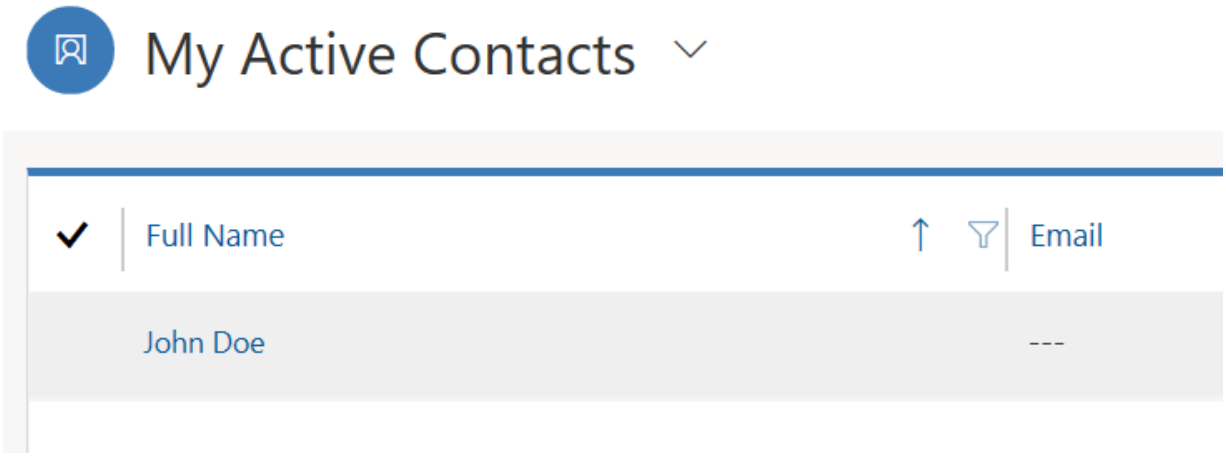
1. Start the application
   - Select **Apps** and click to open the **Permit Management** app in a new window.
2. Create new Contact record
   - Select **Contacts**



   - Click **New** from the top menu.
   - Provide **First Name** as **John** and **Last Name** as **Doe**.
   - Click **Save and Close**.



- You should now see the created contact on the **Active Contacts** view.



3. Create new Build Site record
   - Select **Build Sites** from the sitemap.
   - Click **New**.
   - Provide **Street Address**, **City**, **State/Province**, **Zip/Postal Code**, and **Country/Region** as:
     Street Address: One Microsoft Way

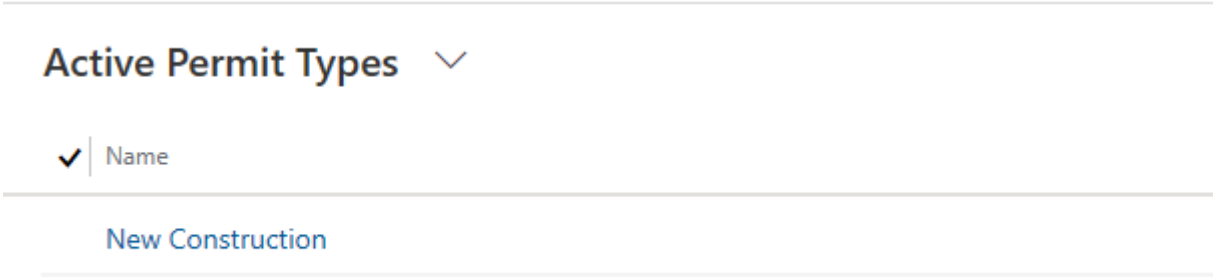
City: Redmond
State/Province: WA

ZIP/Postal Code: 98052

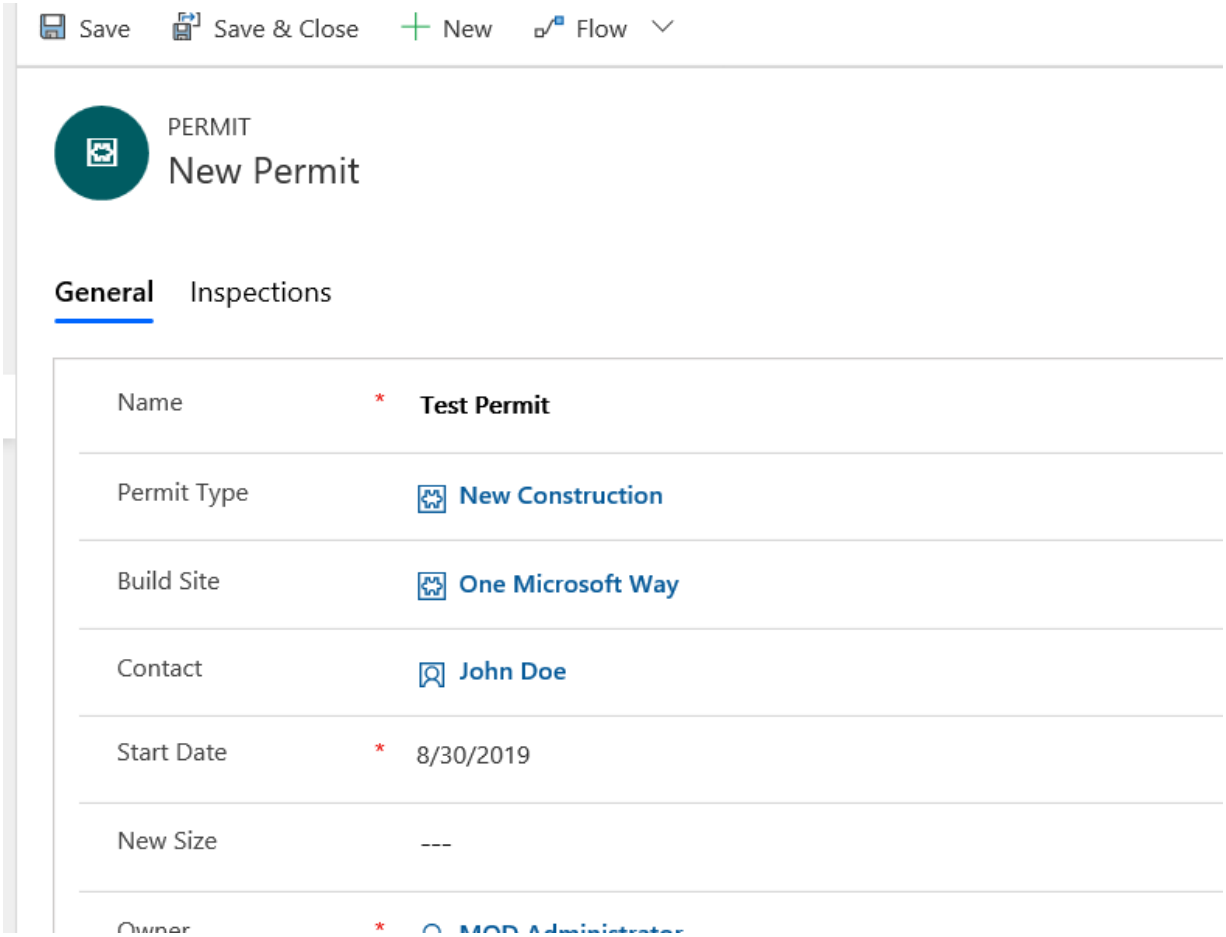
Country/Region: USA

- Click **Save and Close** and this will show the newly created record on the Active Build Sites View.

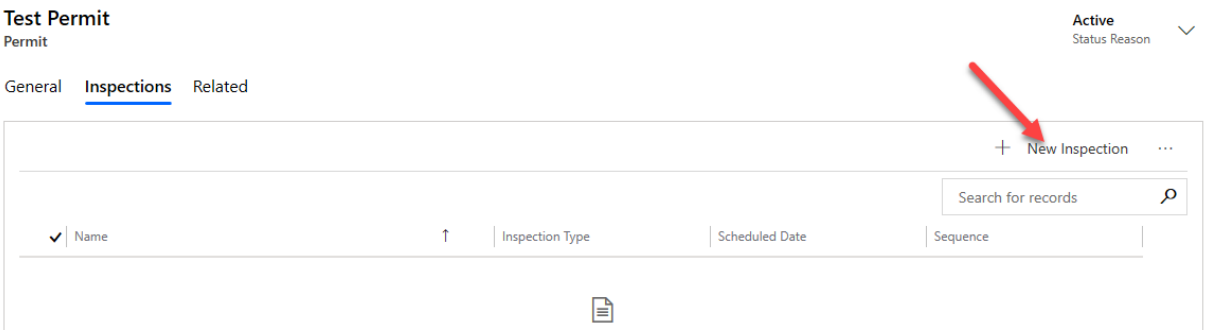
4. Create new Permit Type record
    - Select **Permit Types** from the sitemap.
    - Click **New**.
    - Provide **Name** as **New Construction** and click **Save and Close**. This will create the record and you should be able to see it on the Active Permit Type View.



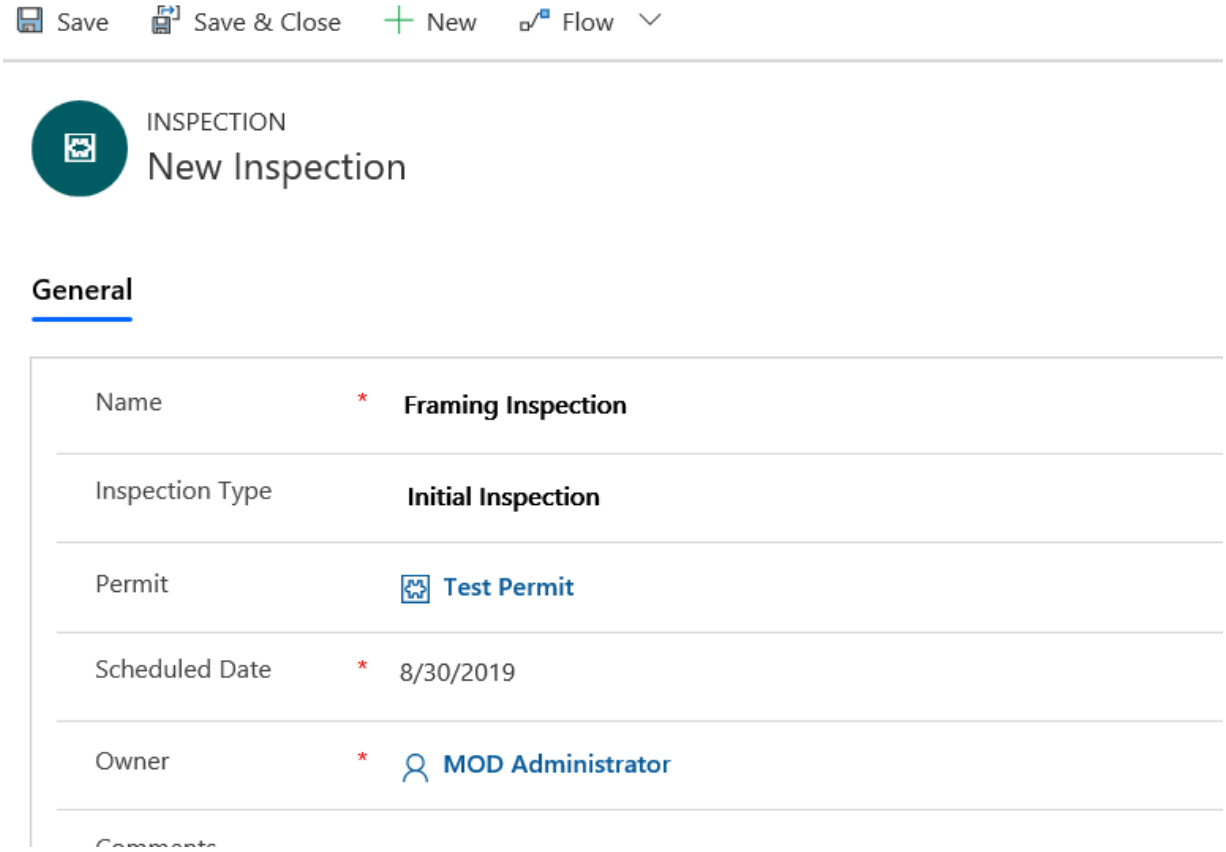
5. Create new Permit record
    - Select **Permits** from the sitemap.
    - Click **New**.
    - Provide **Name** as **Test Permit**, select the **Permit Type**, **Build Site**, and the **Contact** records you created in the previous steps.
    - Select a future date for the **Start Date** and click **Save**.



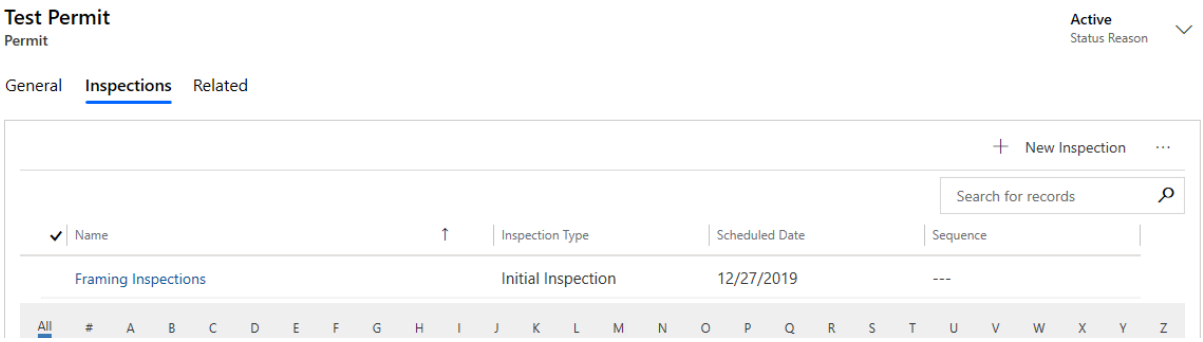
6. Create new Inspection record
    - Go to the **Inspections** tab.
    - Click + **New Inspections**.



- Provide **Name** as **Framing Inspections**, select **Initial Inspection** from the dropdown for **Inspection Type**, and select future date for **Scheduled Date**.
- Click **Save and Close.**



- The **Inspection** record should now show on the **Permit** sub-grid.



7. You may add more test records.

12.3 lab: title: 'Lab 03: Canvas app'

[!NOTE] Effective November 2020:

- Common Data Service has been renamed to Microsoft Dataverse. Learn more
- Some terminology in Microsoft Dataverse has been updated. For example, *entity* is now *table* and *Column* is now *column*. Learn more

This content will be updated soon to reflect the latest terminology.

12.4 Lab 03 – Canvas app

13 Scenario

A regional building department issues and tracks permits for new buildings and updates for remodeling of existing buildings. Throughout this course you will build applications and automation to enable the regional building department to manage the permitting process. This will be an end-to-end solution which will help you understand the overall process flow.

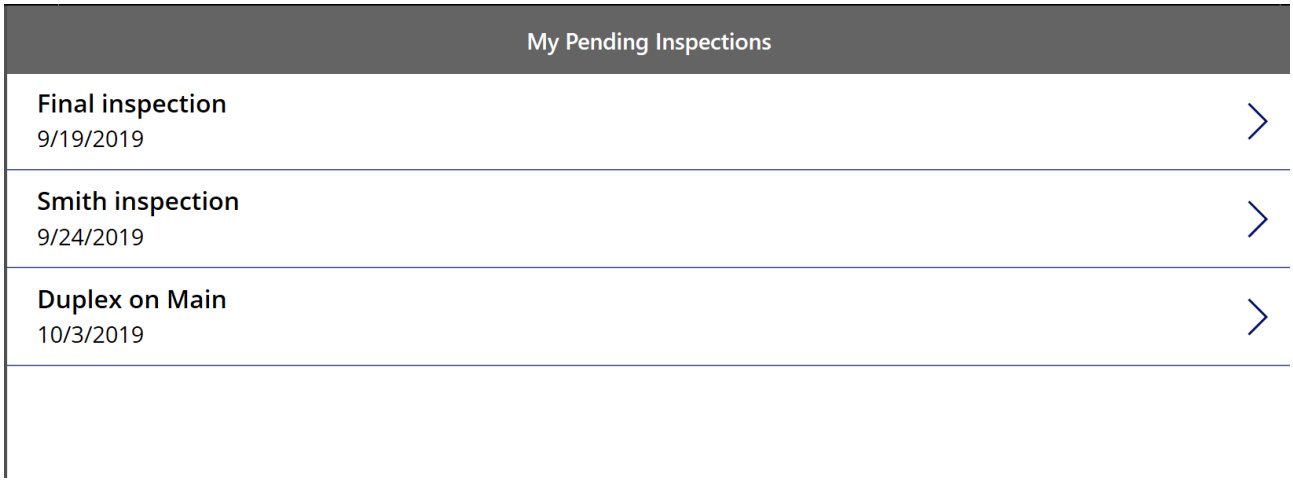
As we continue to build our solution, we will now design a Power Apps canvas app that the inspectors will use in the Column on their mobile devices. Canvas apps are low code apps that can be designed for a tablet or mobile phone layouts. You will build a two-screen canvas app that allows inspectors quickly access and process the inspections.

14 High-level lab steps

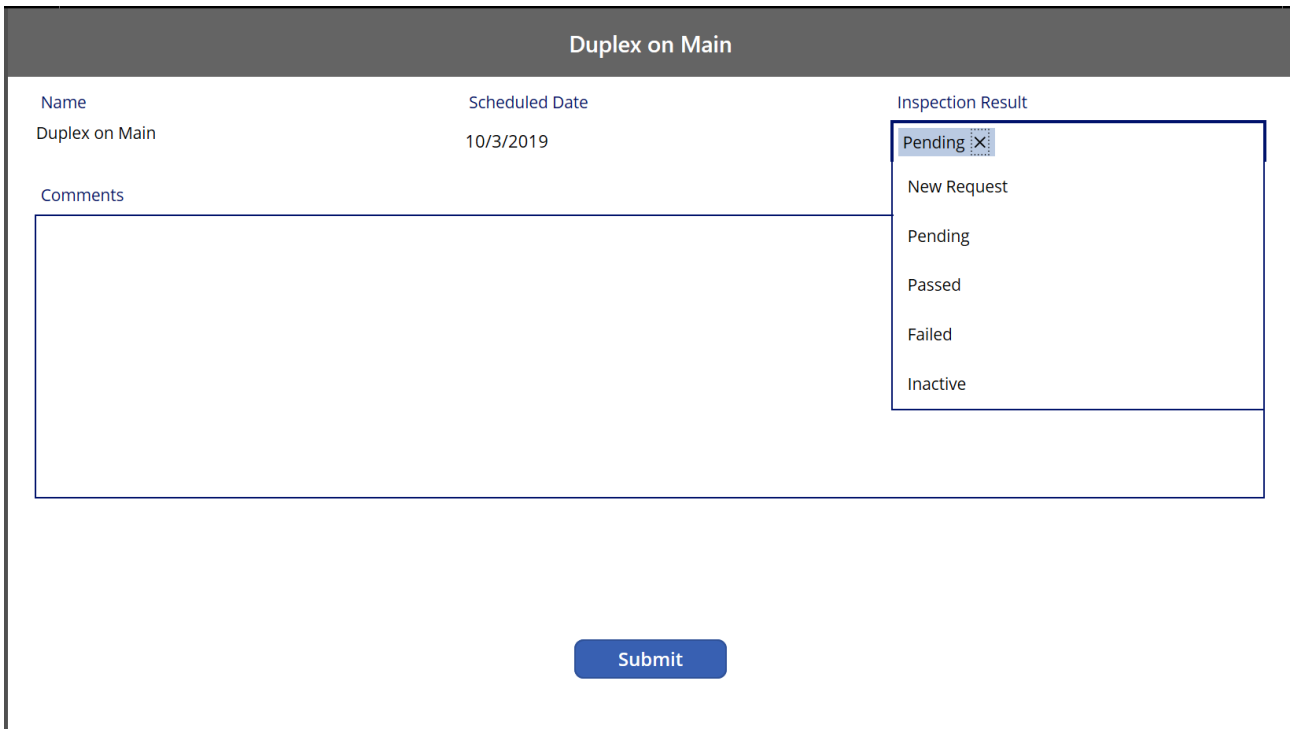
We will follow the below schema to design the canvas app:

- Create the app using the tablet form factor
- Connect to Microsoft Dataverse as a data source
- Configure a gallery control to show the pending inspections
- Use a Microsoft Dataverse view to populate the gallery
- Configure a detail page with inspection info
- Handle saving the inspection results to Microsoft Dataverse
- Export the solution with the data model and apps and import it to the "Production" environment

This is the first screen in the application to show all Pending Inspections for the logged in Inspector.



This second screen lets the inspector update the selected Inspection.



14.1 Things to consider before you begin

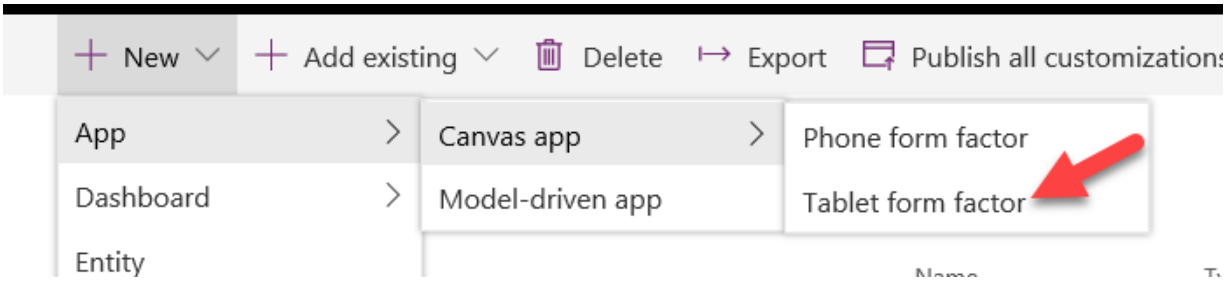
- What would an inspector need quick access to while in the field?
- How do we move our solution from the development to the production environment?
- Remember to continue working in your DEVELOPMENT environment. We'll move everything to production in Exercise 2 of this lab.

15 Exercise #1: Create Canvas App

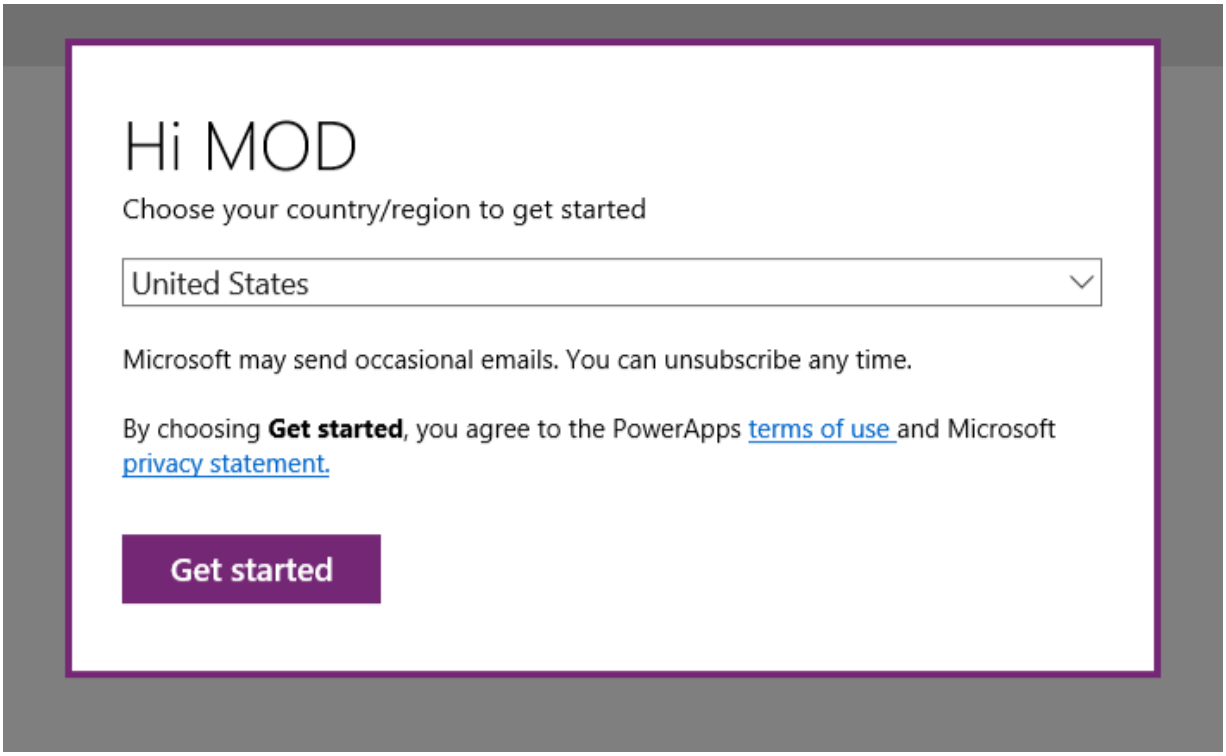
Objective: In this exercise, you will create a canvas app.

15.1 Task #1: Create Canvas App

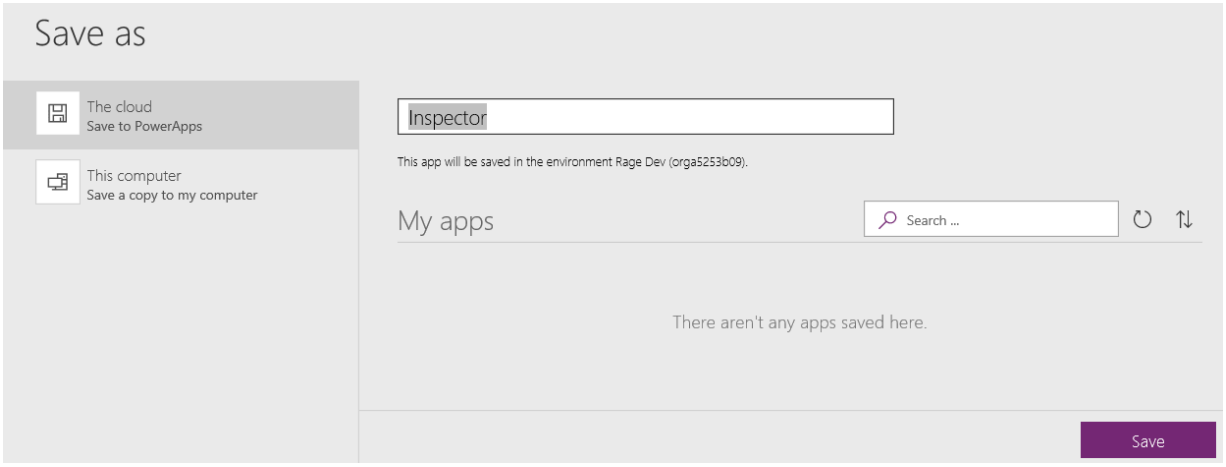
1. Open the Permit Management solution.
   - Sign in to Power Apps maker portal
   - Select your **Dev environment.**
   - Select **Solutions**.
   - Click to open the **Permit Management** solution.
2. Create new canvas application
   - Click + **New** and select **App | Canvas App | Tablet Form Factor**. This will open the App Editor in a New window.



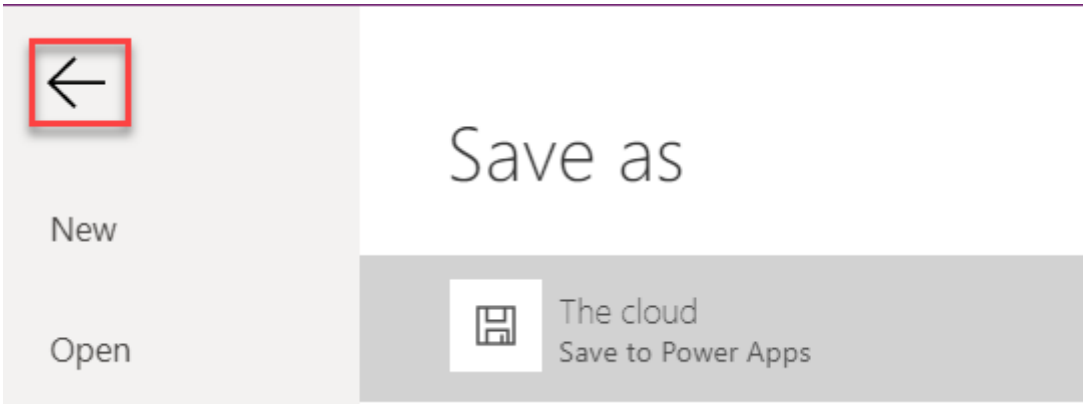
If you are creating your first app, this will ask you to set the Country/region for the app. Click **Get Started.**



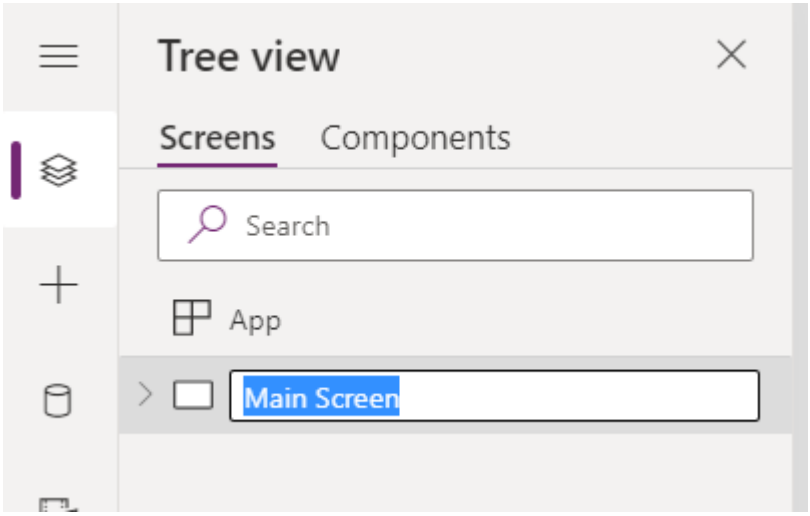
- Click **File** and select **Save As**.
- Select **The Cloud**, enter **Inspector** for Name, and click **Save**. This will make sure that the changes are not removed if the app closes unexpectedly.



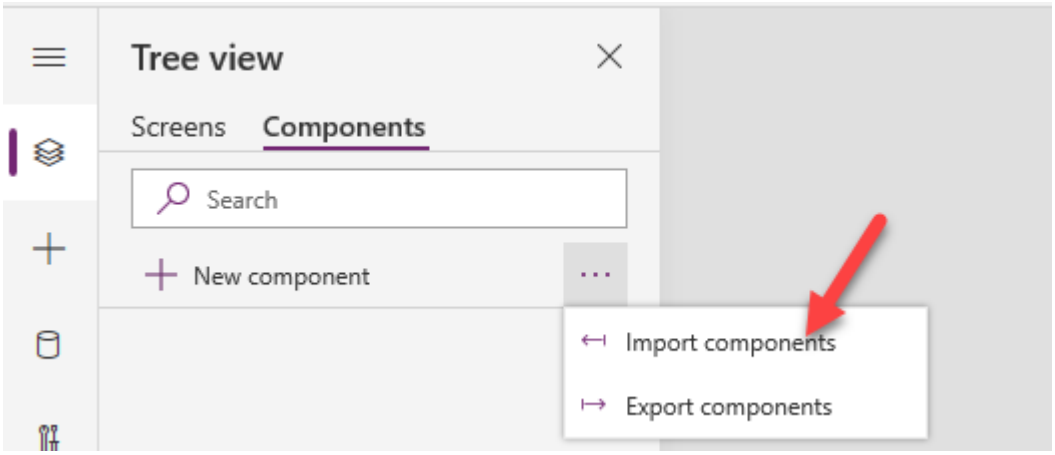
- Click on the app designer button.



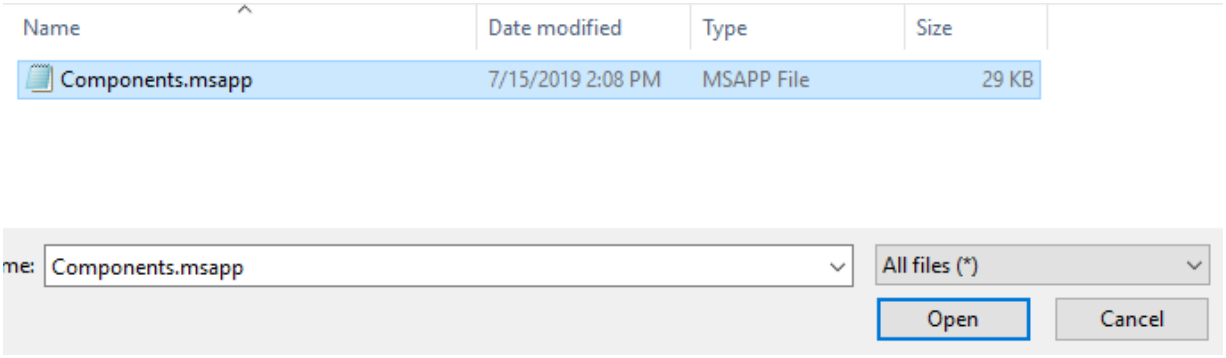
3. Rename Screen1.
    - Go to the **Tree View** and double click on **Screen1**.
    - Rename it **Main Screen** and press **Enter.**



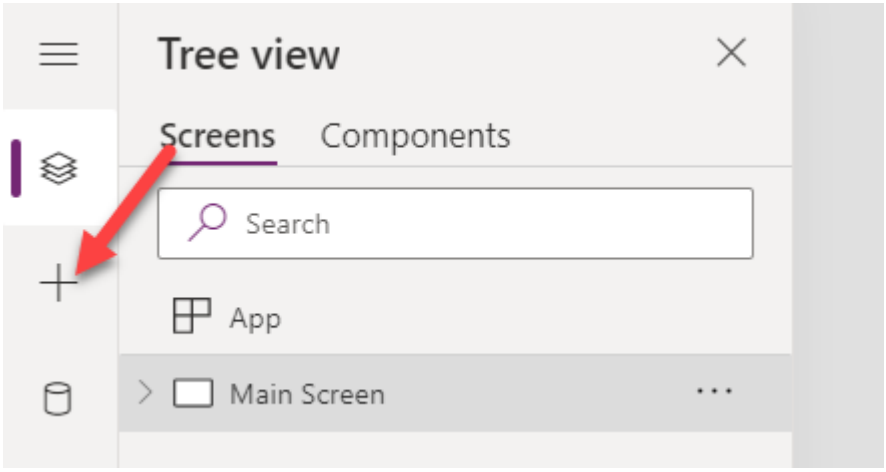
4. Import Component.
    - Select the **Components** tab.
    - Click on the **... (Component Options)** button and select Import **Components**.



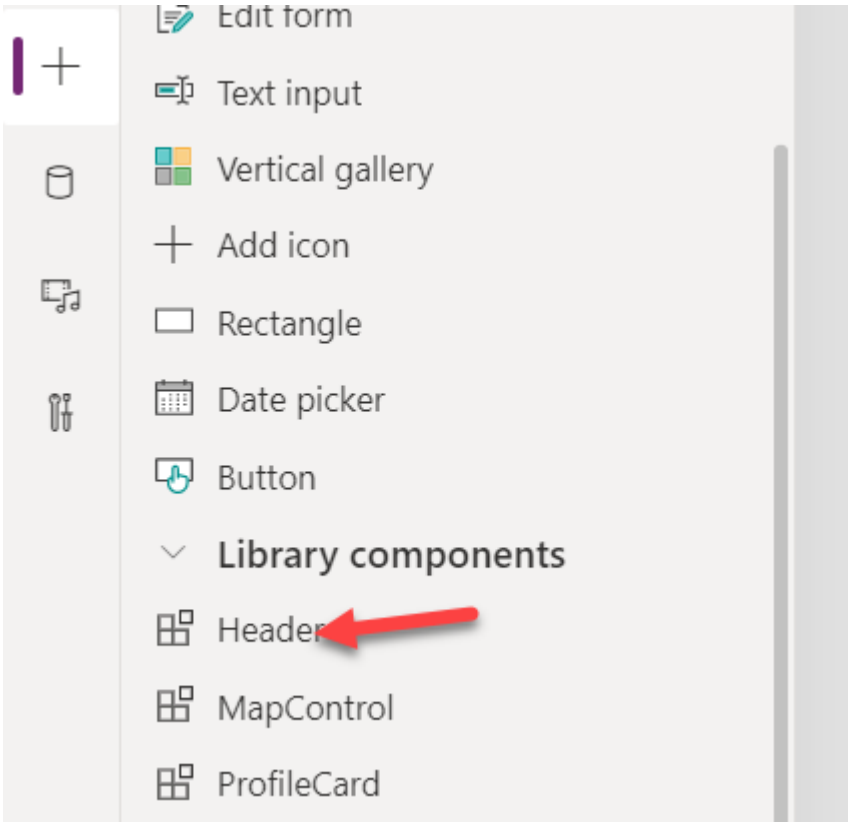
    - Click **Upload File**.
    - Browser to the lab resources folder, select the **Components** file and click **Open**.



5. Add the Header component to the MainScreen.
    - Select the **Screens** tab.
    - Click + Insert.

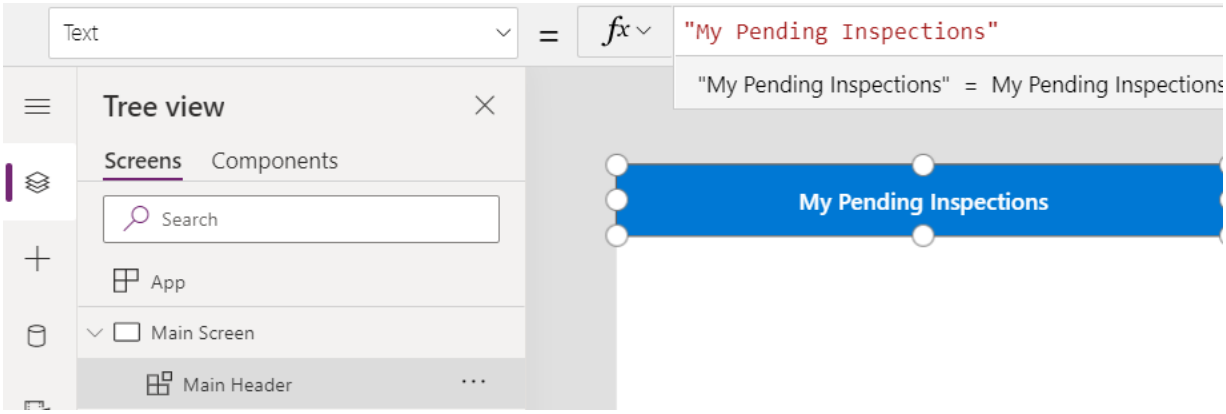


- Click to expand **Library components** and select **Header**.



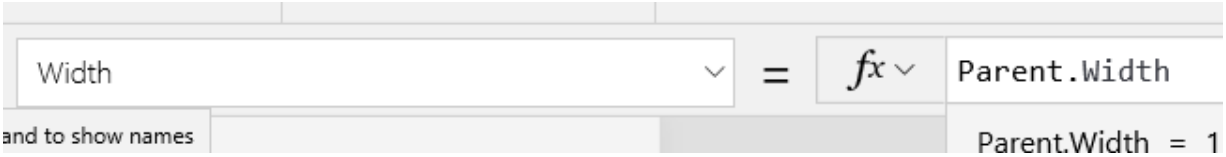
- Select **Tree View**.
- Rename **Header_1** to **Main Header** by double click on Header_1.

6. Change the Main Header properties
   - Select **Main Header**.
   - Change the **Text** Column of the **Main Header** to **My Pending Inspections**. This can be done by selecting "Text" property in the dropdown below top menu. Make sure that you have selected the Main Header control while doing this step.



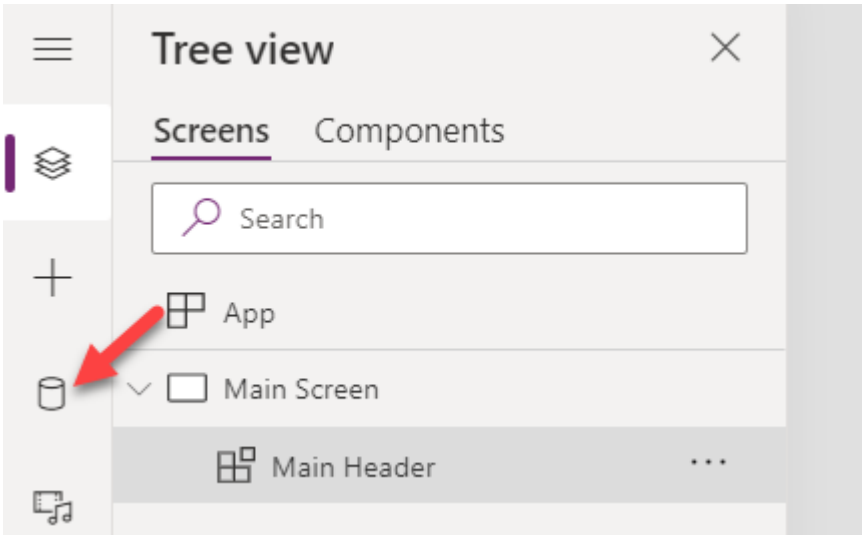
- Change the **Width** vale of the **Main Header** to the formula below.

```
Parent.Width
```

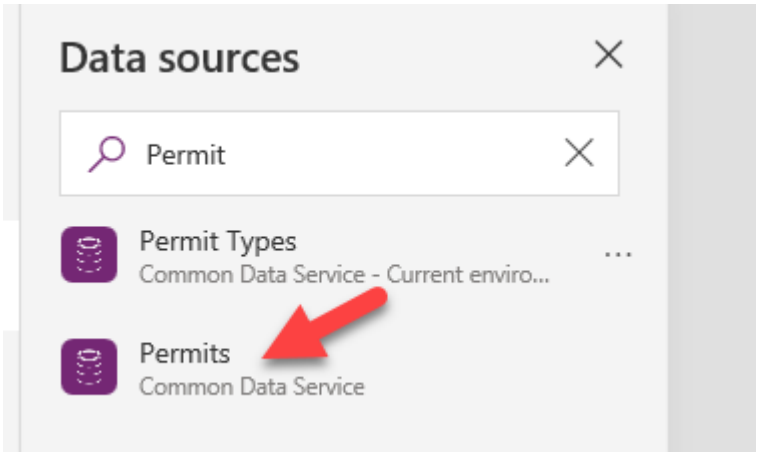


15.2 Task #2: Add Data Source

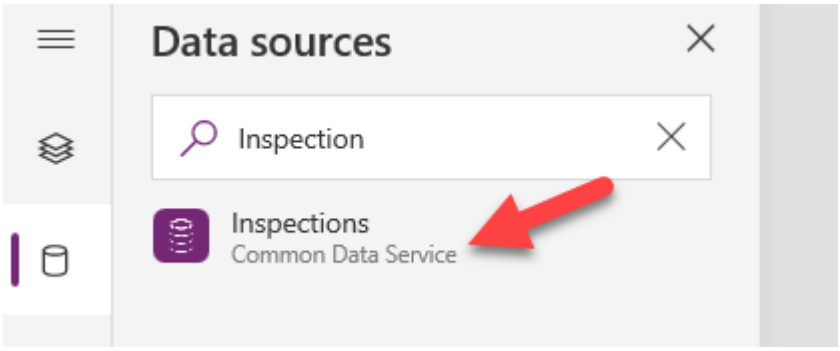
1. Add Permit and Inspection as data source.
   - Select the **Data Sources** tab.



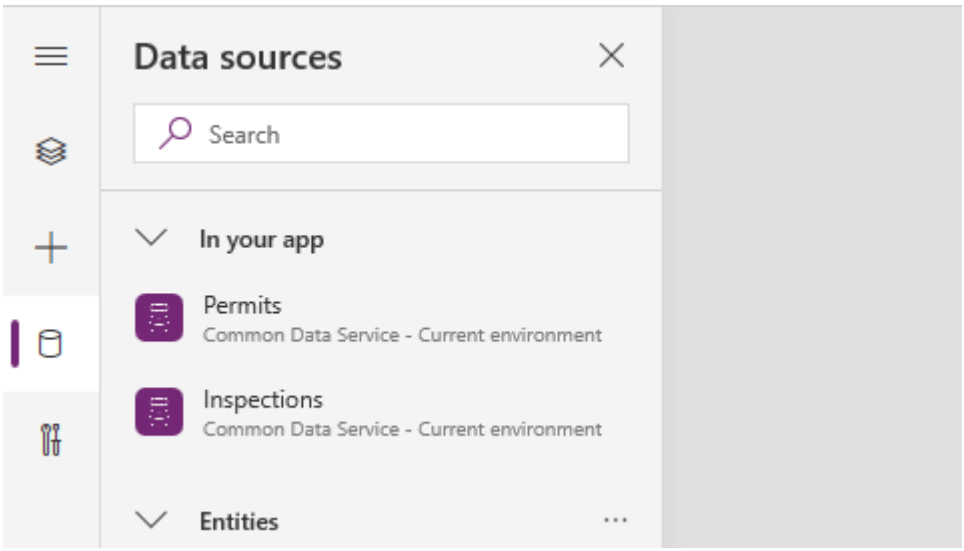
   - Search for **Permit** and select **Permits**.



- Search for **Inspection** and select **Inspections**.

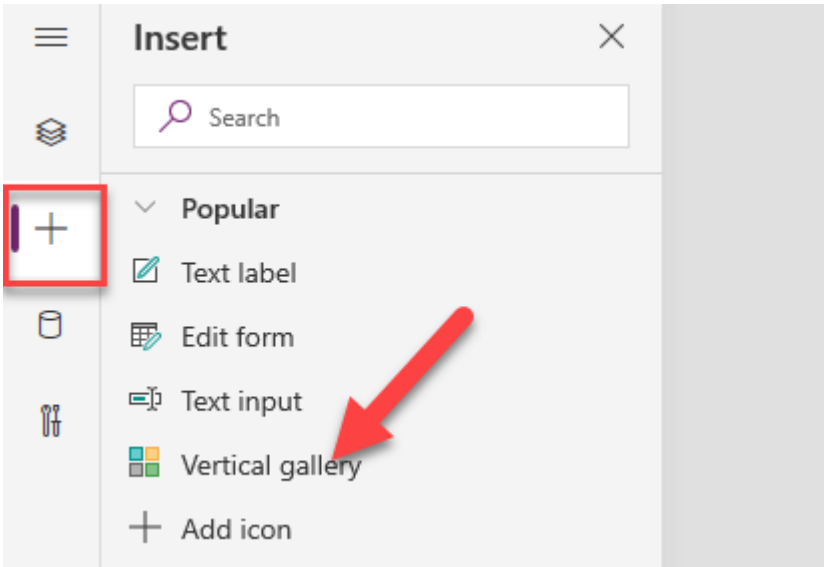


- You should now have both **Permits** and **Inspections** in your app.

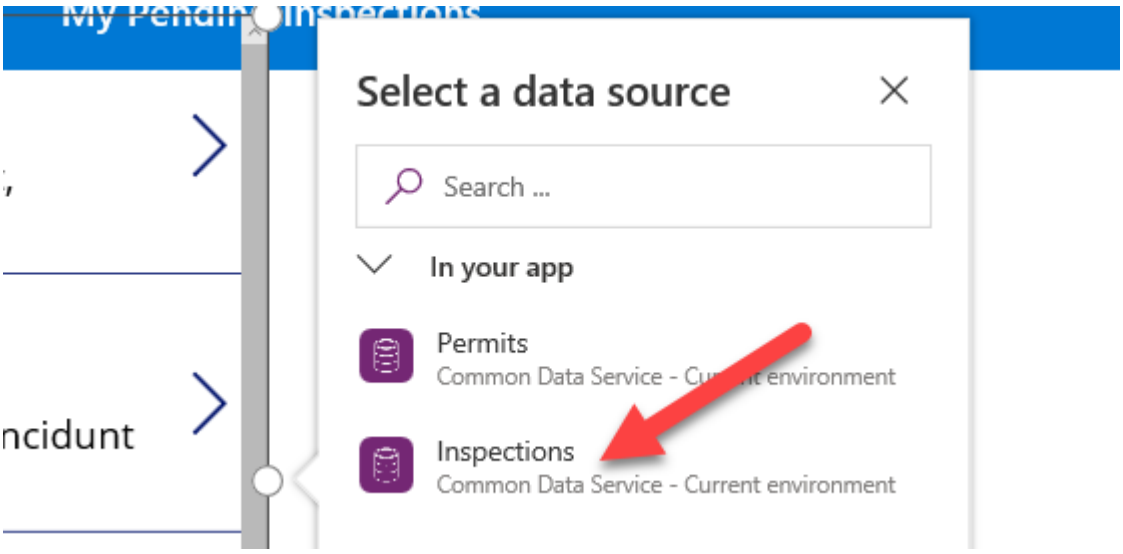


15.3 Task #3: Add Inspection Gallery

1. Add Gallery
   - Click + **Insert** and select **Vertical Gallery**. Vertical Gallery will be added to the MainScreen.

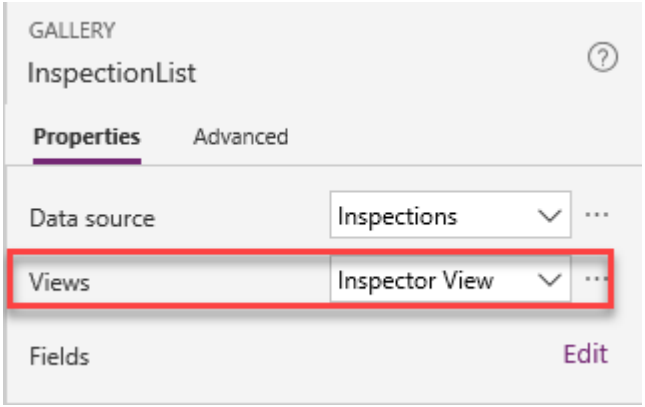


   - Select **Inspections** for **Data Source**. When Inspection is selected, this will automatically pick the Columns and show them in the gallery items.

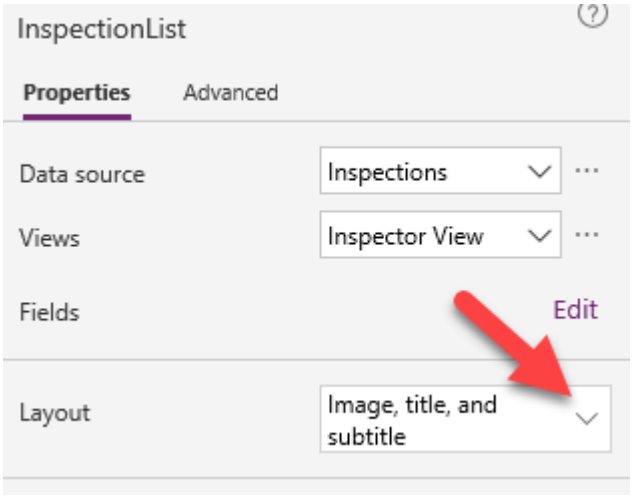


- Select the **Tree View** tab.
- Rename **Gallery_1** to **Inspection List** by double click on Gallery_1.

2. Select the inspector view
   - Make sure you have the **Inspection List** control selected.
   - Go to the **Properties** pane and select **Inspector View** for **View**.



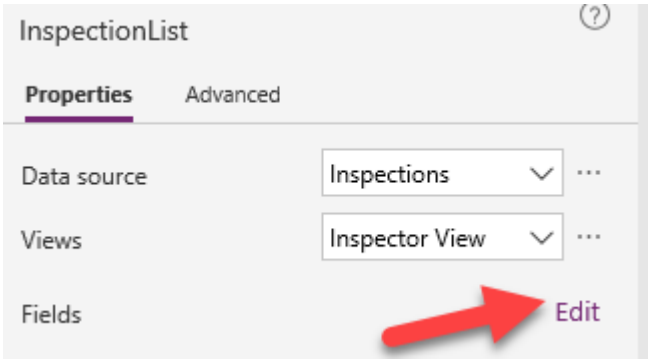
3. Change the **Inspection List** control layout
   - Go to the **Properties** pane and click on the **Layout** dropdown.



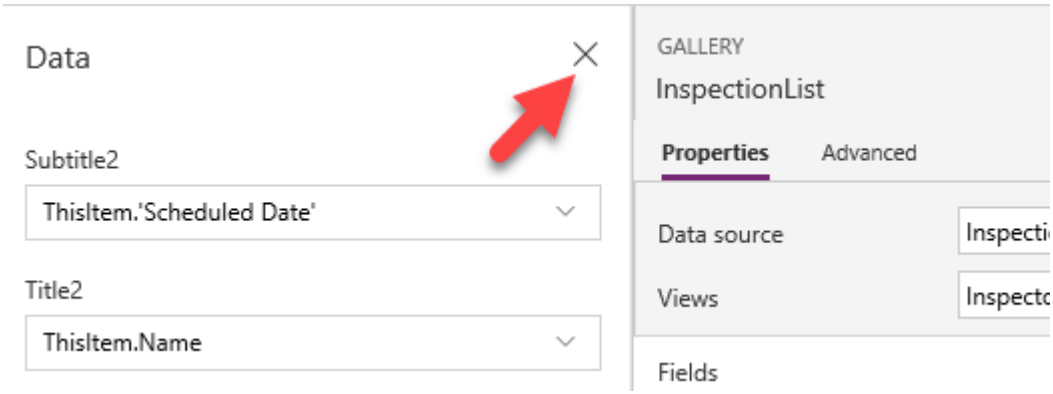
   - Select **Title and Subtitle**.

4. Verify the selected Columns.

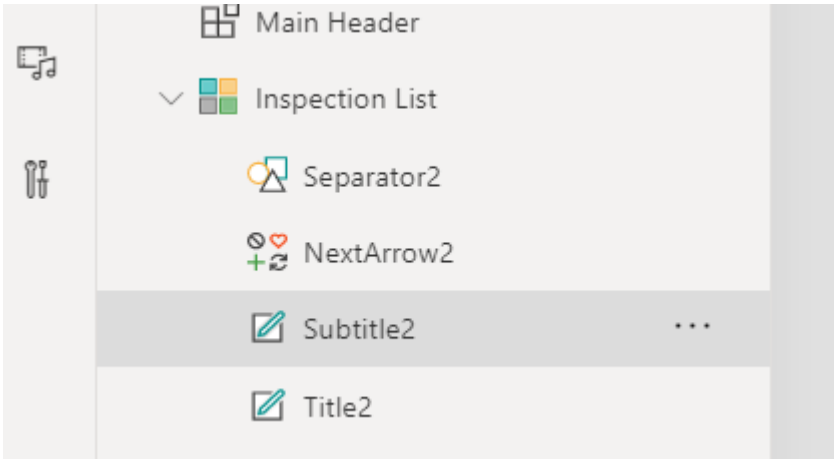
- Go to the **Properties** pane and click on the **Edit Columns**.



- Confirm that **Scheduled Date** is selected for **Subtitle** and **Name** is selected for **Tile**. Close the **Data** pane.



5. Change date time to date only.
   - Expand the **Inspection List** and select **Subtitle**.



   - Change the **Text** property of the control to the formula below.

     ```
     DateValue(Text(ThisItem.'Scheduled Date'),"en")
     ```

6. Resize the Gallery
   - Select the **Inspection List** gallery.
   - Select **Width** property from the formula dropdown and enter the formula below.

     ```
     Parent.Width
     ```

   - Select **Height** property and set it to the formula below.

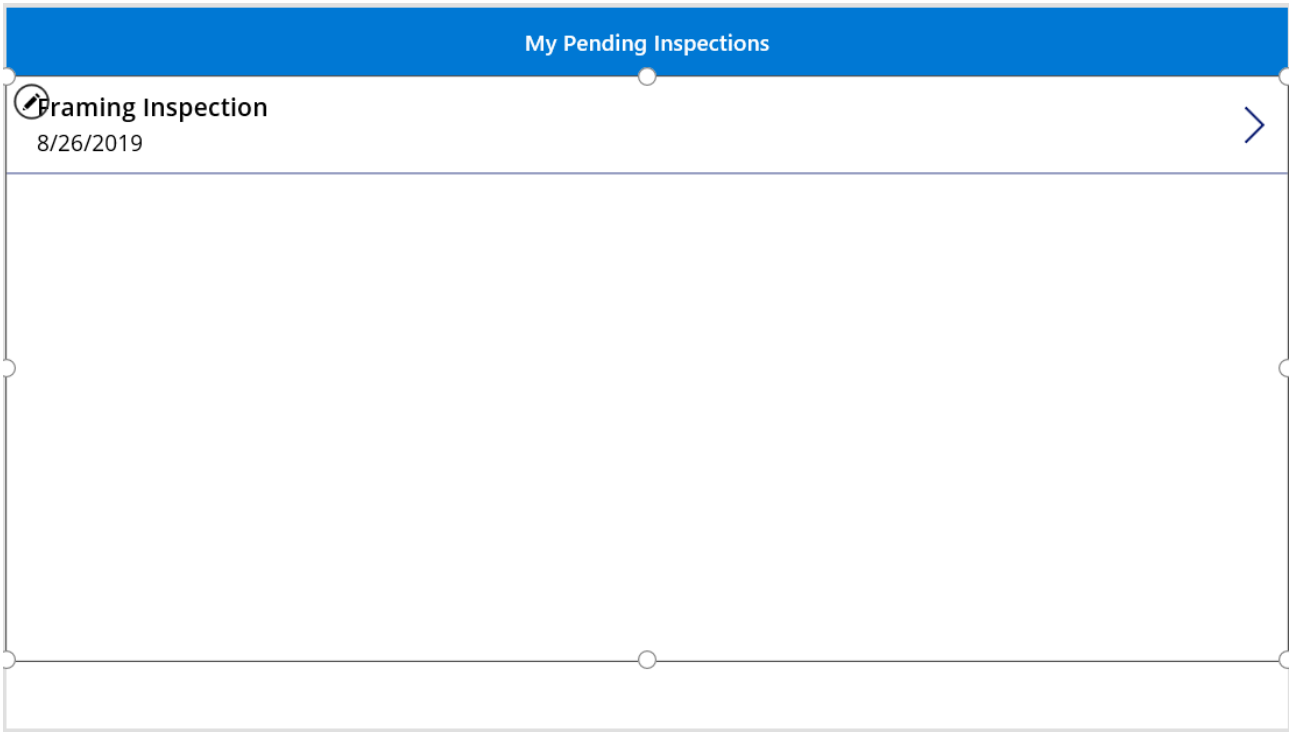
     ```
     Parent.Height - ('Main Header'.Height*2)
     ```

   - Select the Y property from the dropdown and set it to formula below.

```
'Main Header'.Height
```

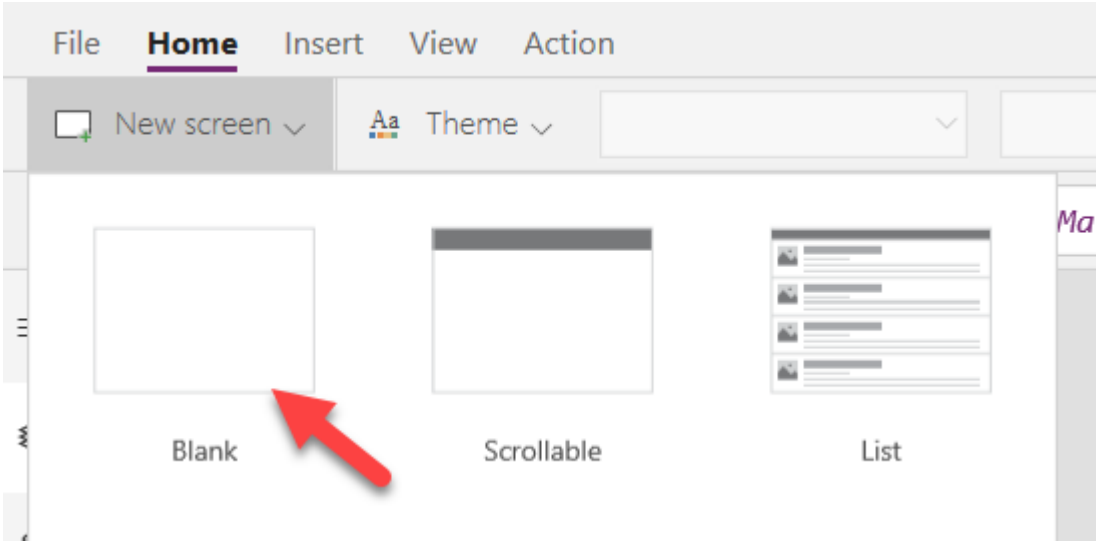
- Select the X property from the dropdown and set it to formula below.

```
'Main Header'.X
```

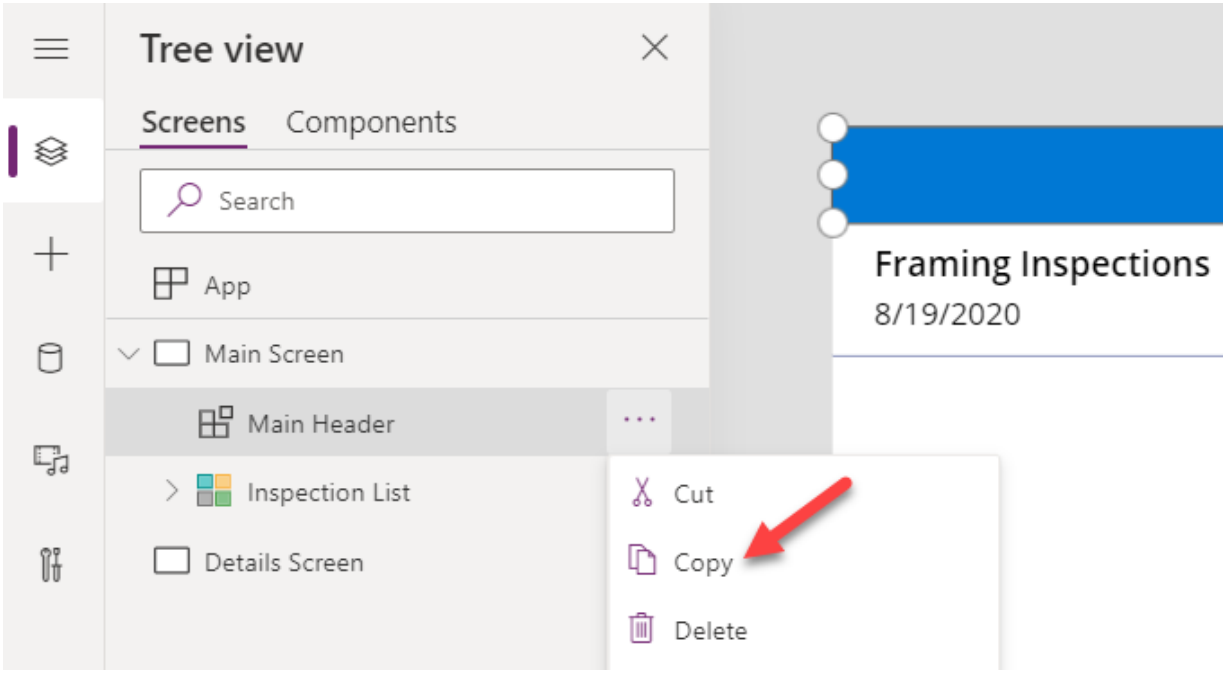


15.4 Task #4: Add Inspection Details Screen

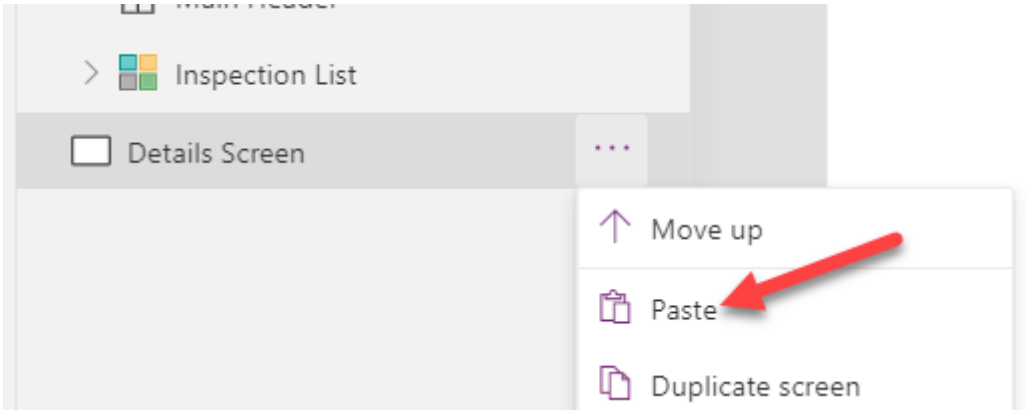
1. Add new screen named Details Screen
   - Click **New Screen** and select **Blank**.



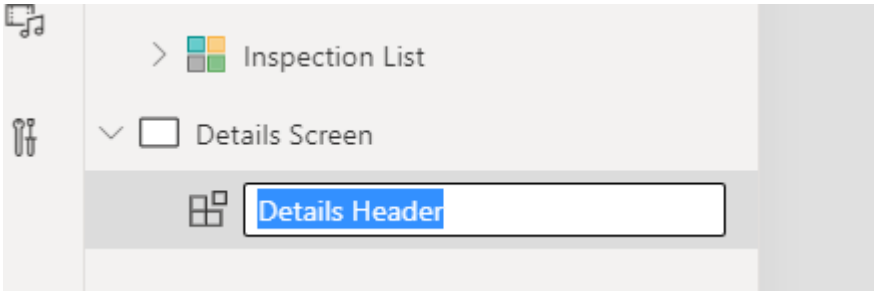
   - Rename the new screen **Details Screen** by double clicking on the control in Tree View.
2. Add Header to the Details Screen and edit it
   - Go to the **Main Screen** and copy the **Main Header**.



- Go to the **Details Screen** and paste the **Header**.

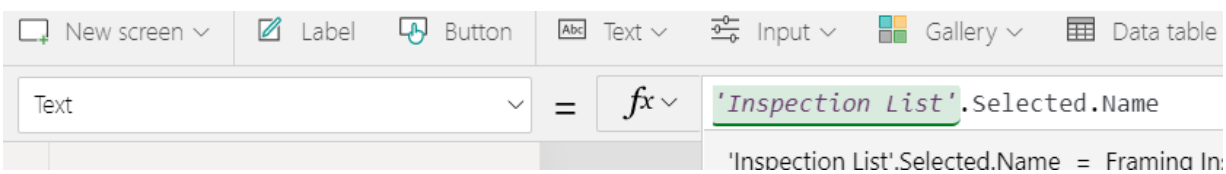


- Rename the Header you **Details Header** by double clicking on the control in Tree View.



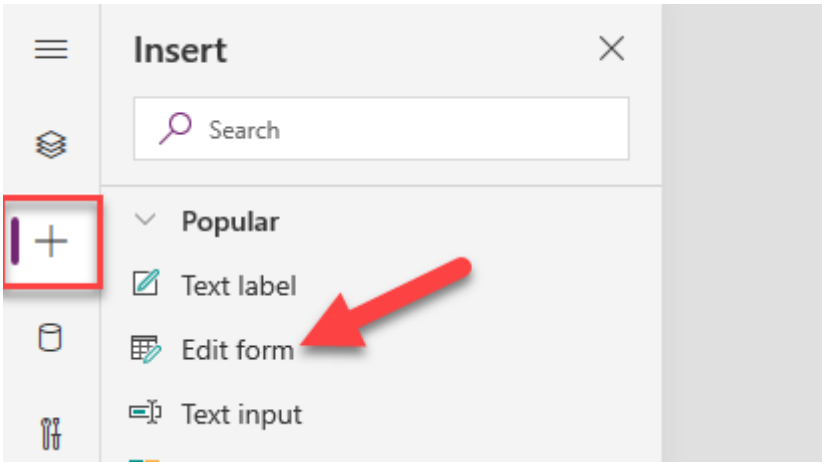
- Select te **Y** property of the **Details Header** and set to **0**.
- Select the **Text** property of the **Details Header** and set it to formula below.

```
'Inspection List'.Selected.Name
```



3. Add Form to the Details Screen.
   - Select the **Details Screen**.

- Click + **Insert** and select **Edit Form**.



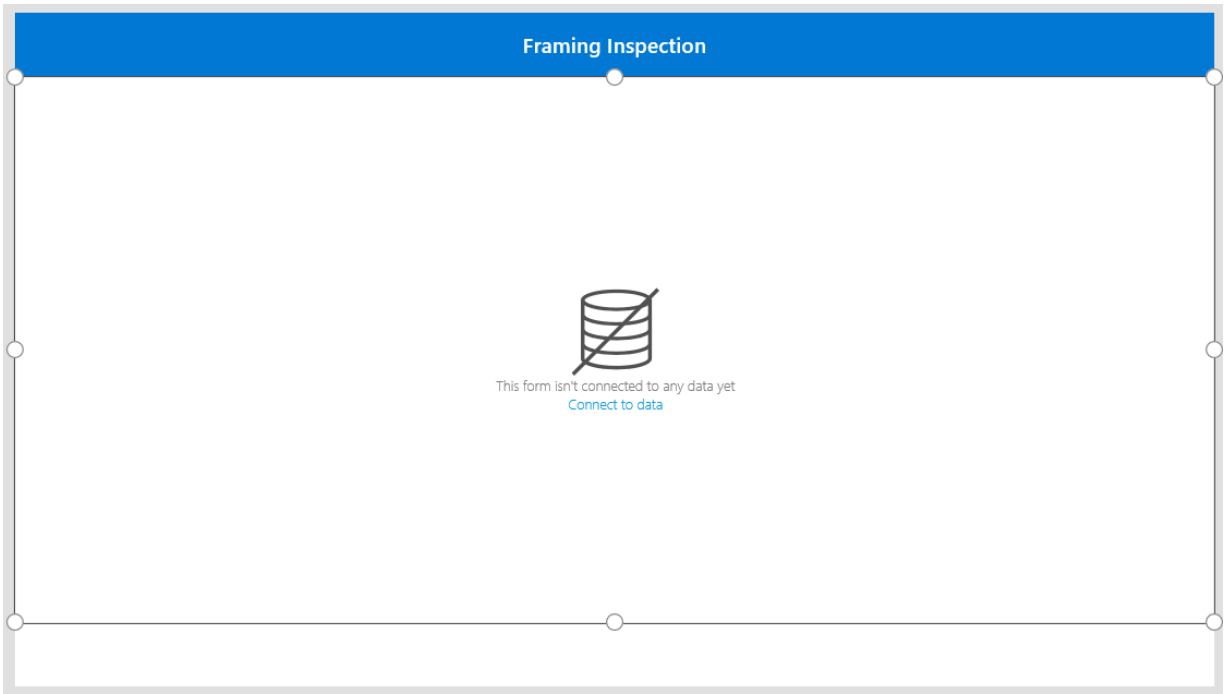
  - Select the **Tree View** tab.
  - Rename the form **Inspection Form**.
  - Resize the Edit form as:
    - Select the **InspectionForm**.
    - Select **Width** property from the formula dropdown and enter the formula below.

      ```
      Parent.Width
      ```
  - Select **Height** property and set it to the formula below.

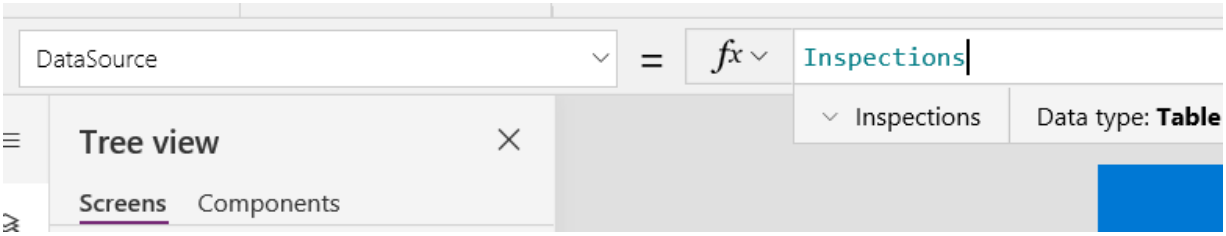
    ```
    Parent.Height - ('Details Header'.Height*2)
    ```
  - Select the Y property from the dropdown and set it to formula below.

    ```
    'Details Header'.Height
    ```
- Select the X property from the dropdown and set it to formula below.

  ```
  'Details Header'.X
  ```
- The form should now look like the image below.

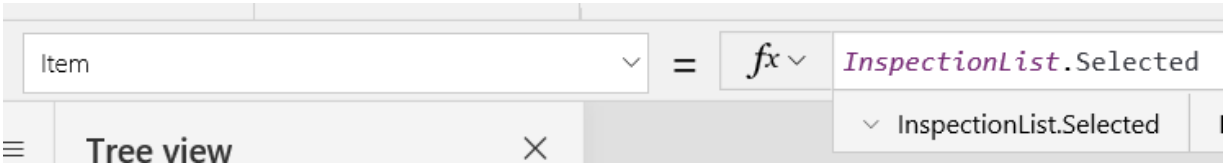


4. Set the **Inspection Form** data source
   - Select the **Inspection Form** and select the DataSource as **Inspections** Table.

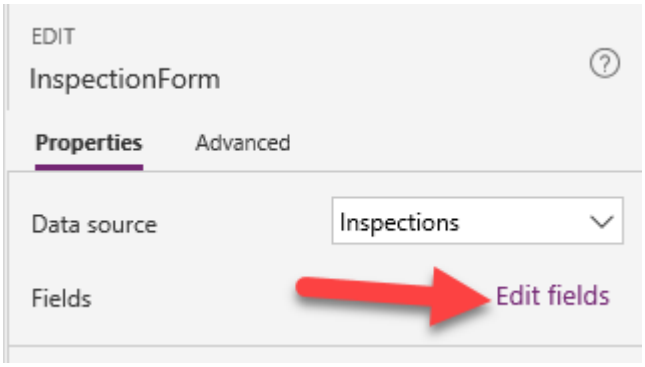


- Set the Item value to the formula below.

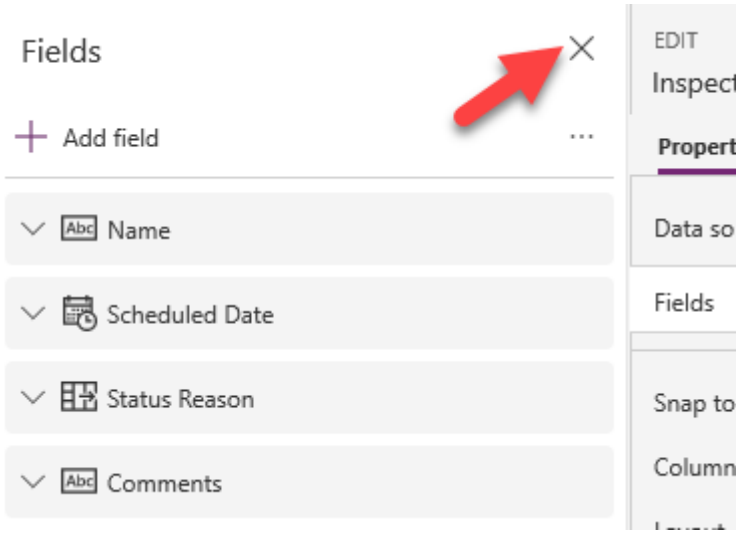
```
'Inspection List'.Selected
```



5. Edit Inspection Form Columns. This adds the data cards for Columns by default, but you can add/remove the data cards as:
   - Select the **Inspection Form**.
   - Go to the **Properties** pane and click **Edit Columns**.

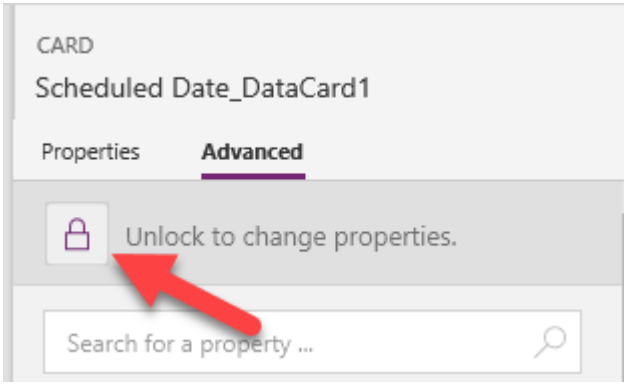


   - Click **Add Column**.
   - Select **Status Reason**, and **Comments**.
   - Click **Add**.
   - The Columns should be ordered as shown in the image below. You can drag/drop to rearrange the Columns.

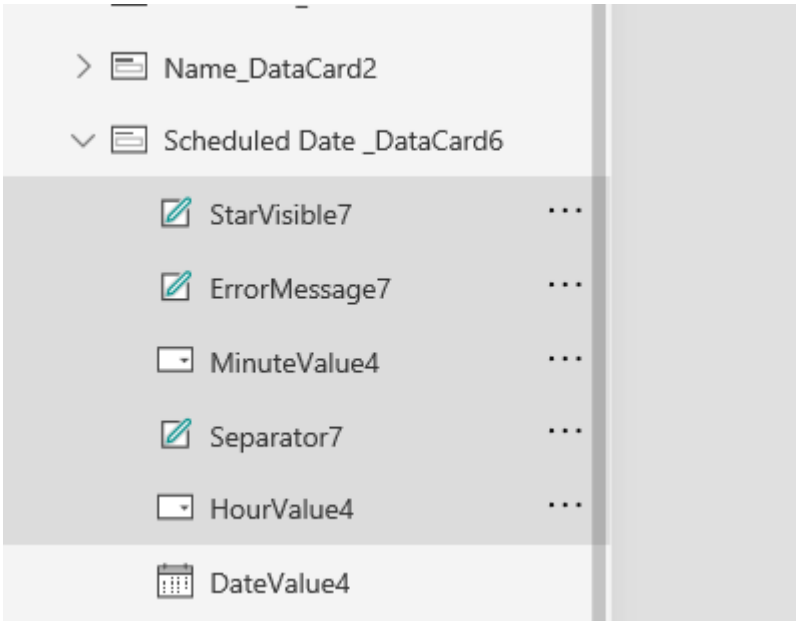


6. Change the Scheduled Date to show date only.
   - Go to the **Tree View** and expand the **Inspection Form**.

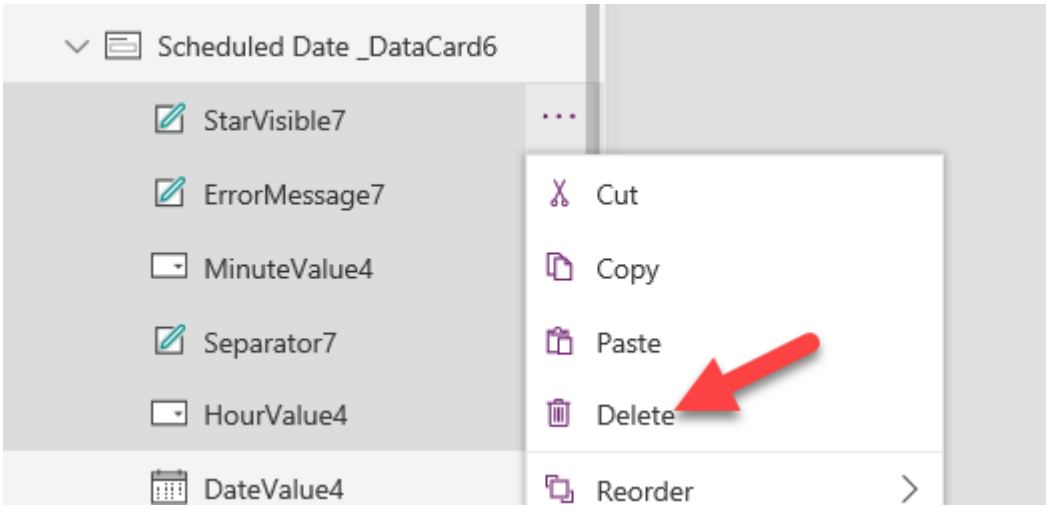
- Select the **Scheduled Date** data card.
- Go to the **Properties** pane and select the **Advanced** tab.
- Click **Unlock to change Properties**.



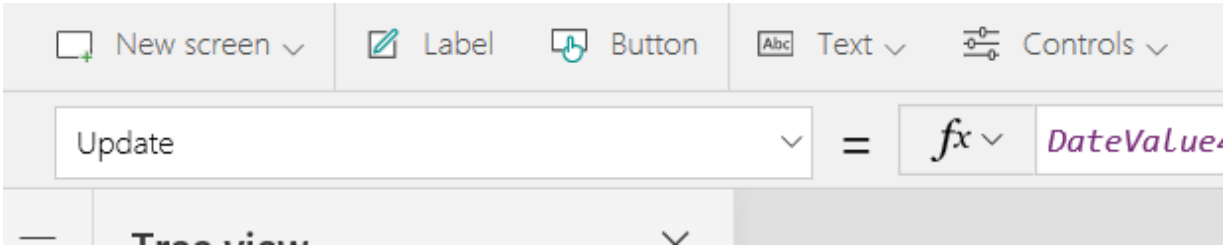
- Expand the **Scheduled Date** card.
- Select **StarVisible**, **ErrorMesage**, **MinuteValue**, **Separator**, and **HourValue**.



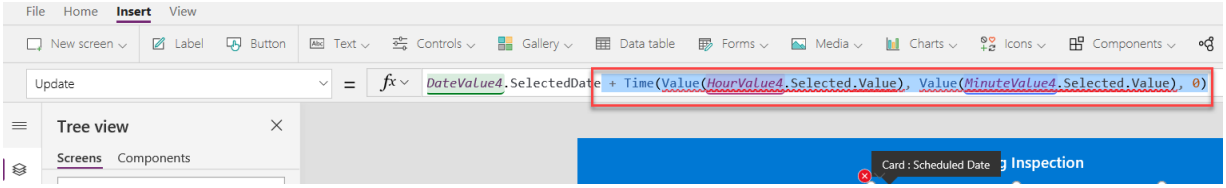
- Delete the selected controls. When the controls are deleted, you will see an error message.



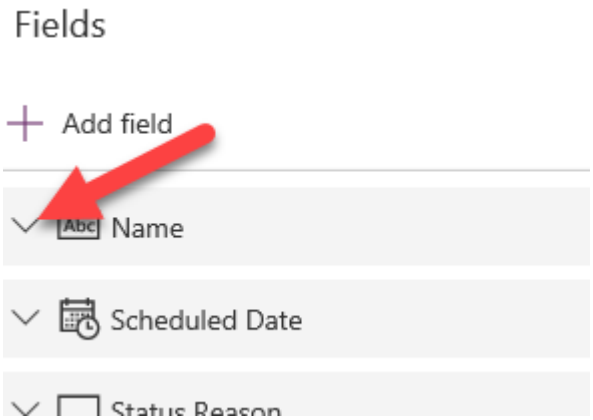
- Select the **Scheduled Date** DataCard.
- Go to formula bar and select **Update**.



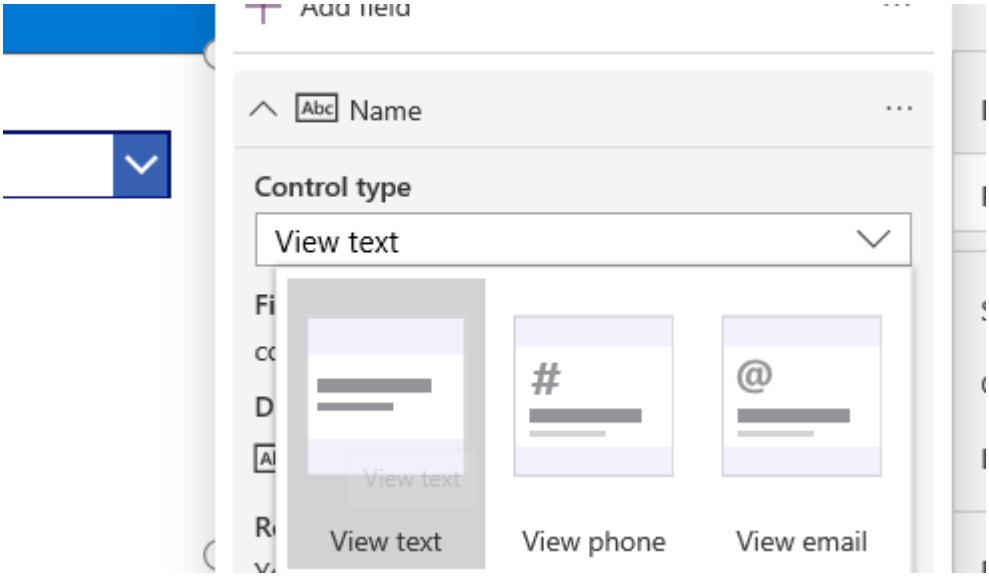
- Remove everything after the **SelectedDate**. This should remove the error message from the app.



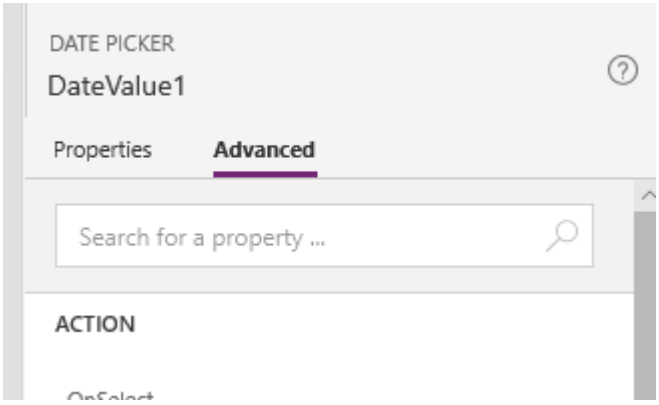
7. Make the Name and Scheduled Date Columns read-only
   - Select the **Inspection Form**.
   - Go to the **Properties** pane and click **Edit Columns**.
   - Expand the **Name** Column.



   - Click on the **Control Type** dropdown and select **View Text**.

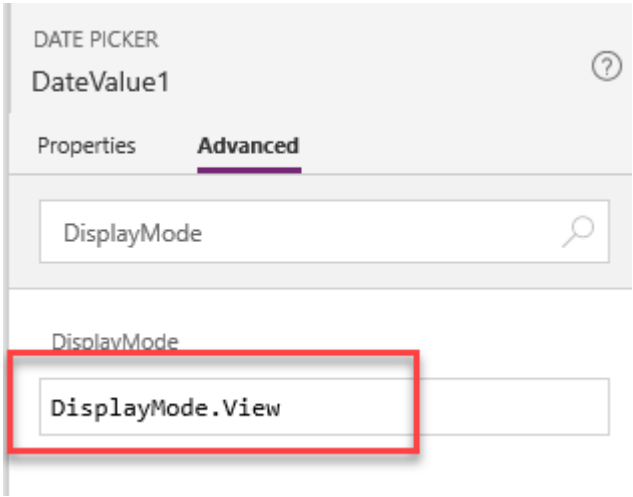


   - Expand the **Scheduled Date** Column. Observe the change.
   - Notice we cannot change this the same way because we've customized it. From the Tree View select **DateValue** control inside the **Scheduled Date** Datacard and go to the **Advanced tab** of the **Properties pane**.

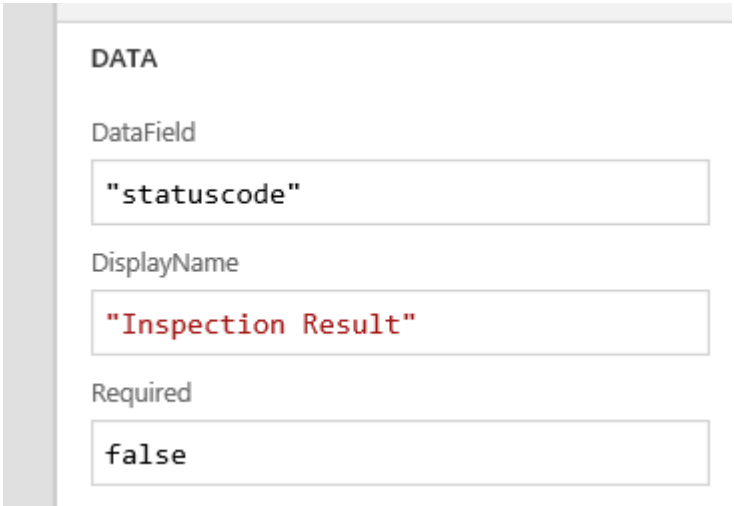


- Search for **DisplayMode** property and remove the existing formula and replace it with the following:

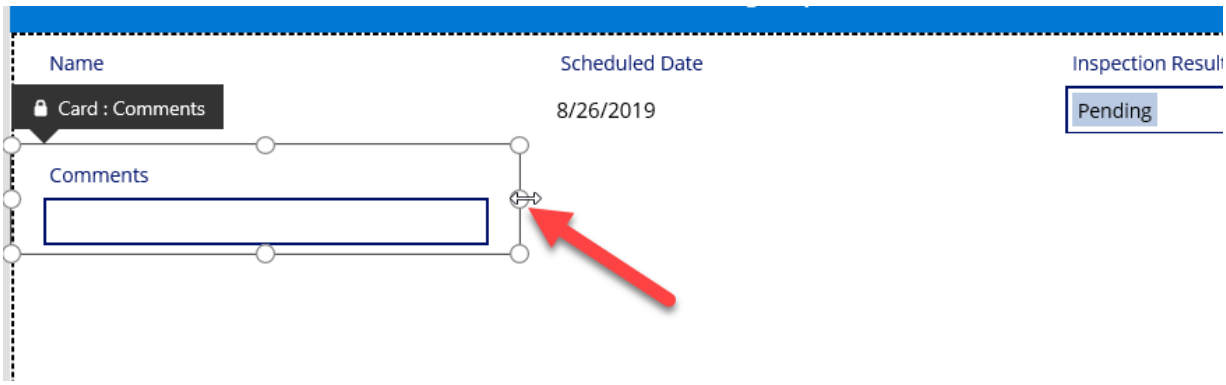
```
DisplayMode.View
```



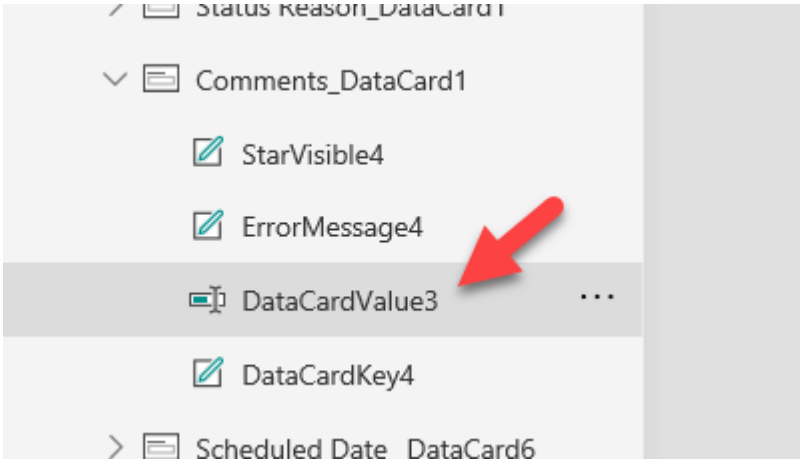
8. Change the Status Reason label.
   - Select the **Status Reason data card**.
   - Go to the **Properties** pane and the Advanced tab, click **Unlock to Change properties.**
   - Change the **DisplayName** to **Inspection Result**.



9. Resize the Comments data card.
   - Select the **Comments** data card.
   - Click and drag the right edge to the far right of the screen.

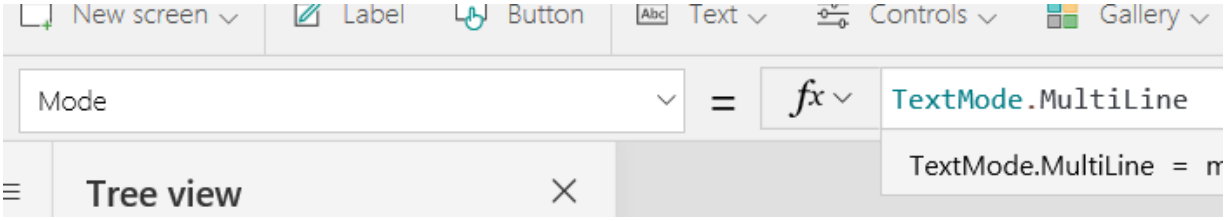


- Go to the **Advanced** tab of **Properties** pane and click **Unlock to change properties**.
- Set the **Height** value to **300**.
- Select the **DataCardValue** control.

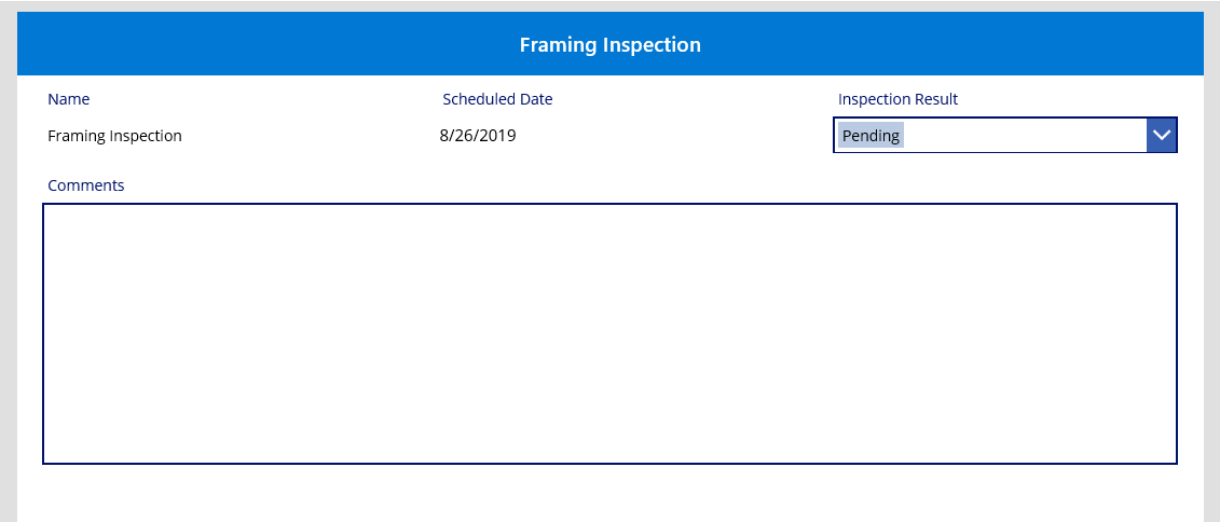


- Set the **Height** value to **300**.
- Change the **Mode** to the formula below.

```
TextMode.MultiLine
```

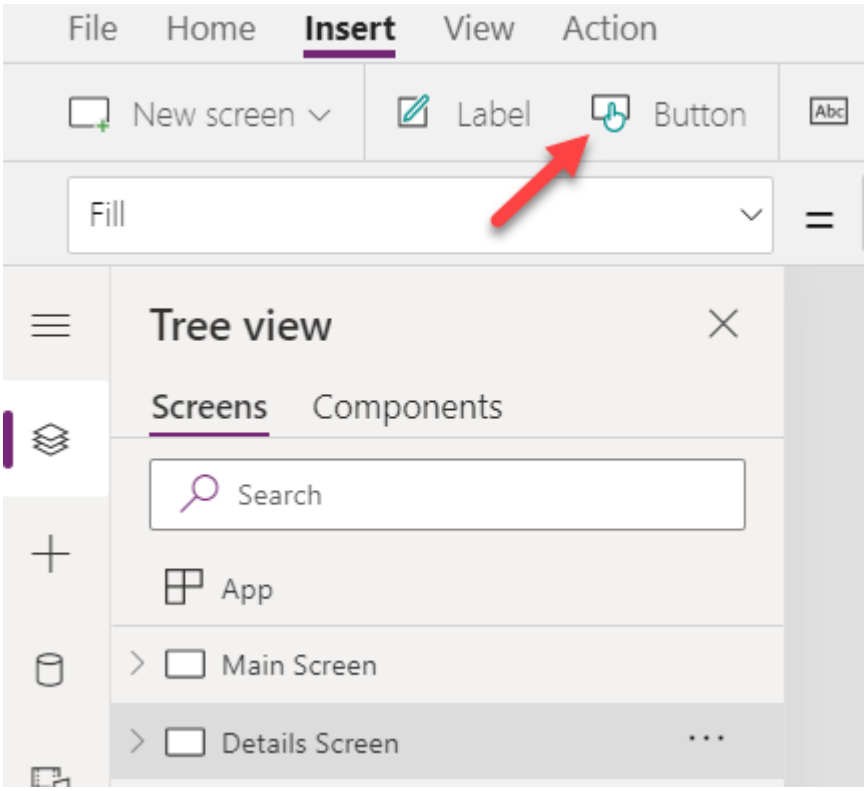


10. Make sure your form looks like the image below. Save your work.

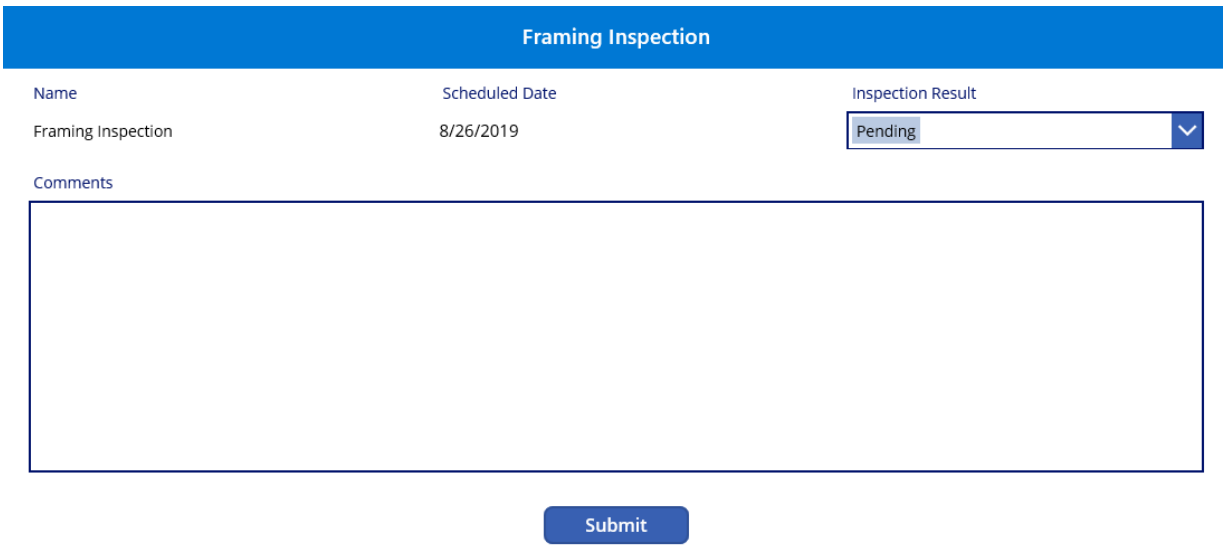


15.5 Task #5: Submit the Inspection Result

1. Add submit button to the details screen.
   - Select the **Details Screen**. Make sure that you have selected the screen and not selected the Edit Form.
   - Go to the **Insert** tab and click **Button**.

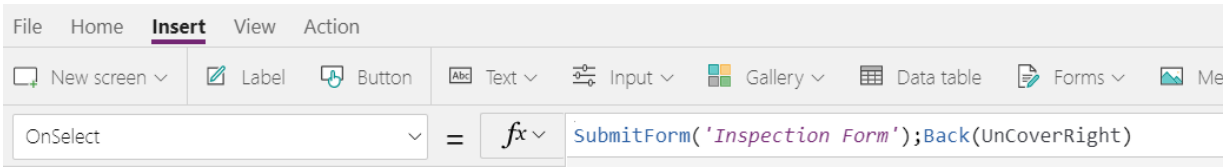


   - Rename the button **Submit Button**.
   - Change the Text value of the button to **Submit**.
   - Place the button below the form through drag and drop.



2. Submit the inspection result.
   - Select the **Submit Button.**
   - Set the **OnSelect** value of the submit button to the formula below. Remove the false expression and update it. This formula will submit the form and then navigate back to the MainScreen.

SubmitForm('Inspection Form');Back(ScreenTransition.UnCoverRight)

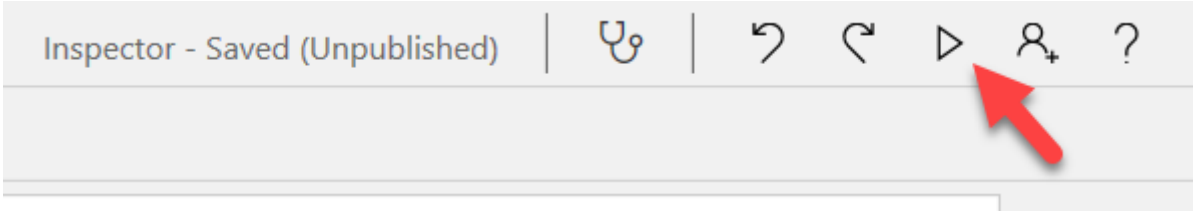


3. Add navigation from the main screen to the details screen.
   - Go to the **Main Screen** and select the **Inspection List**.
   - Set the **OnSelect** property of the **Inspection List** to the formula below. Remove the already existing false expression.

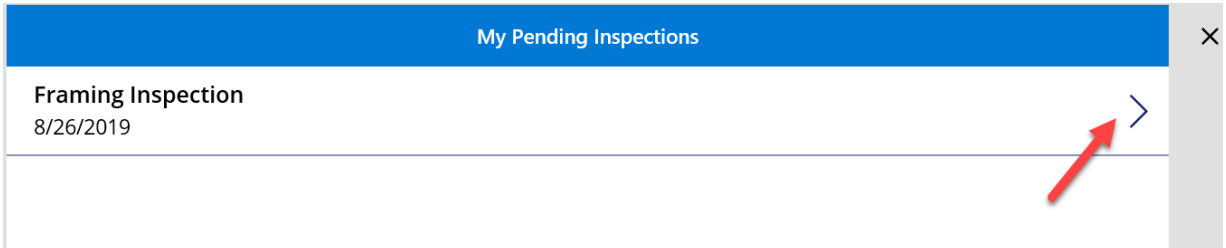
```
Navigate('Details Screen', ScreenTransition.Cover)
```

15.6 Task #6: Test Application

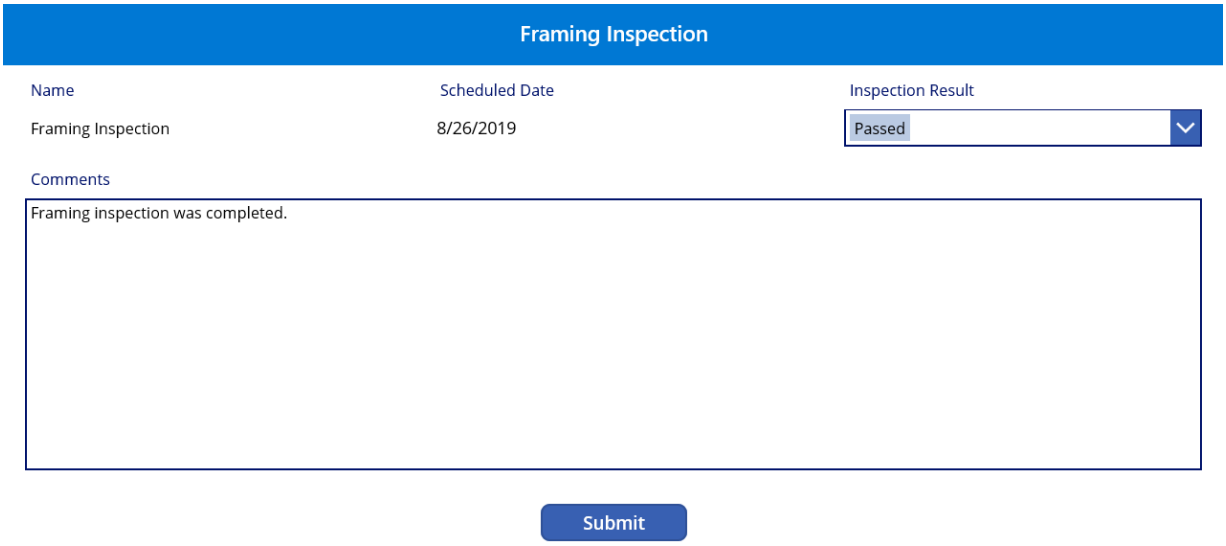
1. Start the application
   - Select the **Main Screen** and click **Preview the App**.



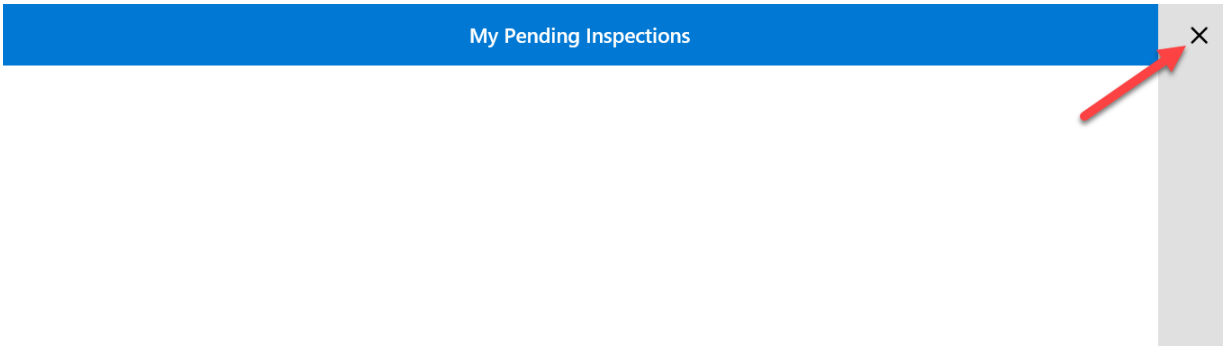
   - The application should load and show at least one inspection. Click on the inspection.



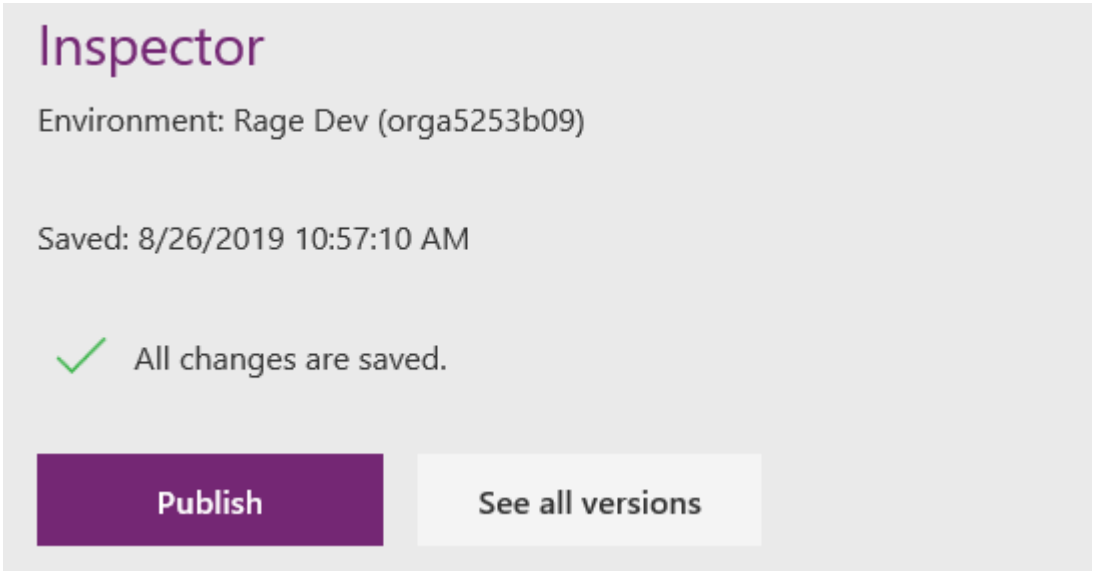
- The application should navigate to the details screen. Change the **Inspection Result** to **Passed**, provide a comment in the textbox as "Framing inspection was completed.", and click **Submit**.



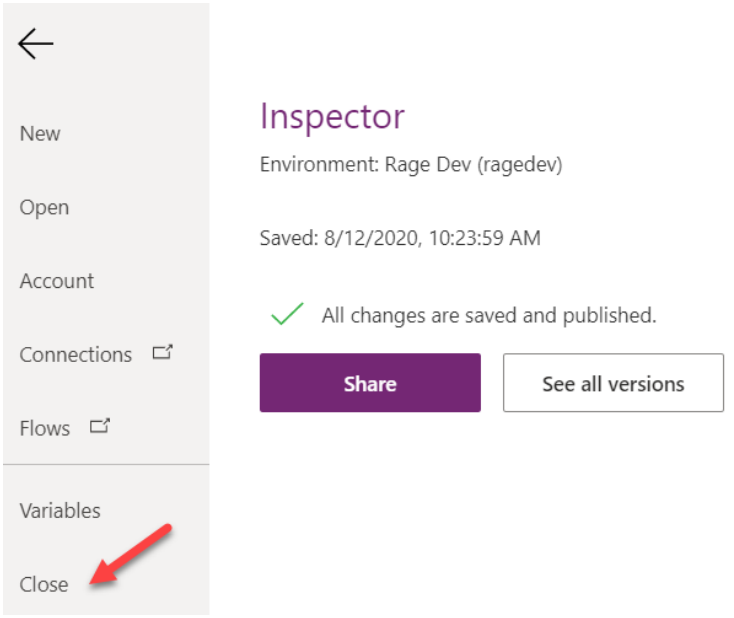
- The inspection should be submitted, and the application should navigate back to the MainScreen. Click Close.



2. Save and publish the application
   - Click **File** and then click **Save**.
   - Click **Publish**.

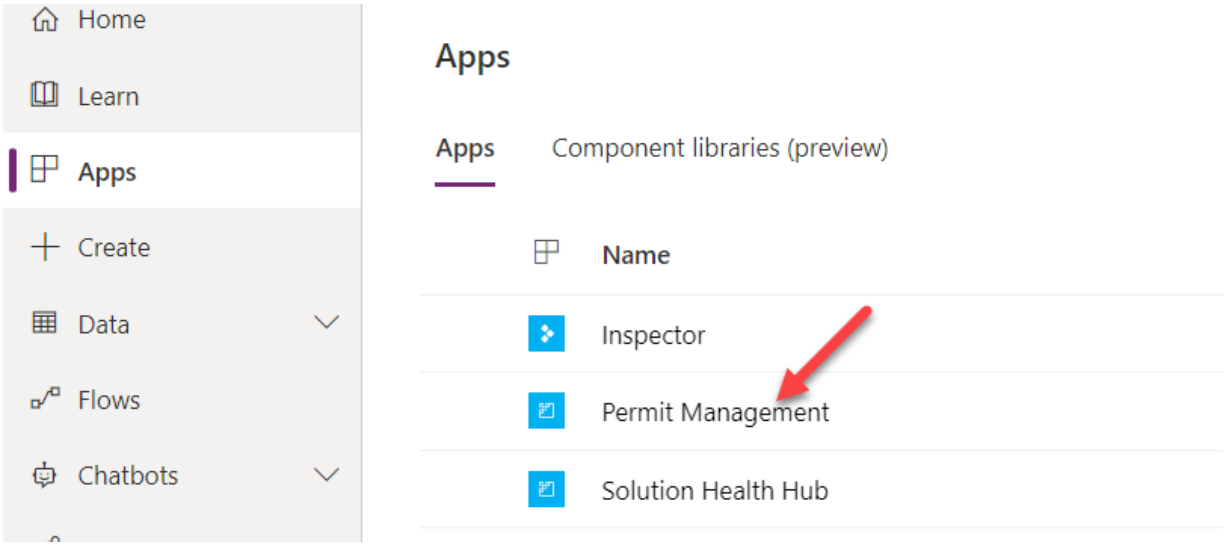


   - Click **Publish this Version**.
   - Click **Close**.

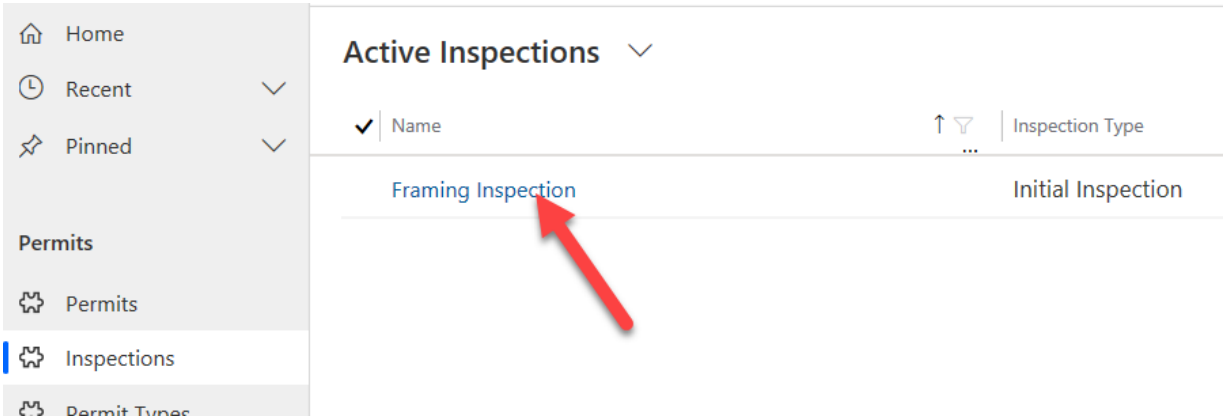


- Close the **Designer** browser window or tab.
- Click **Leave** if prompted when trying to close the browser window.

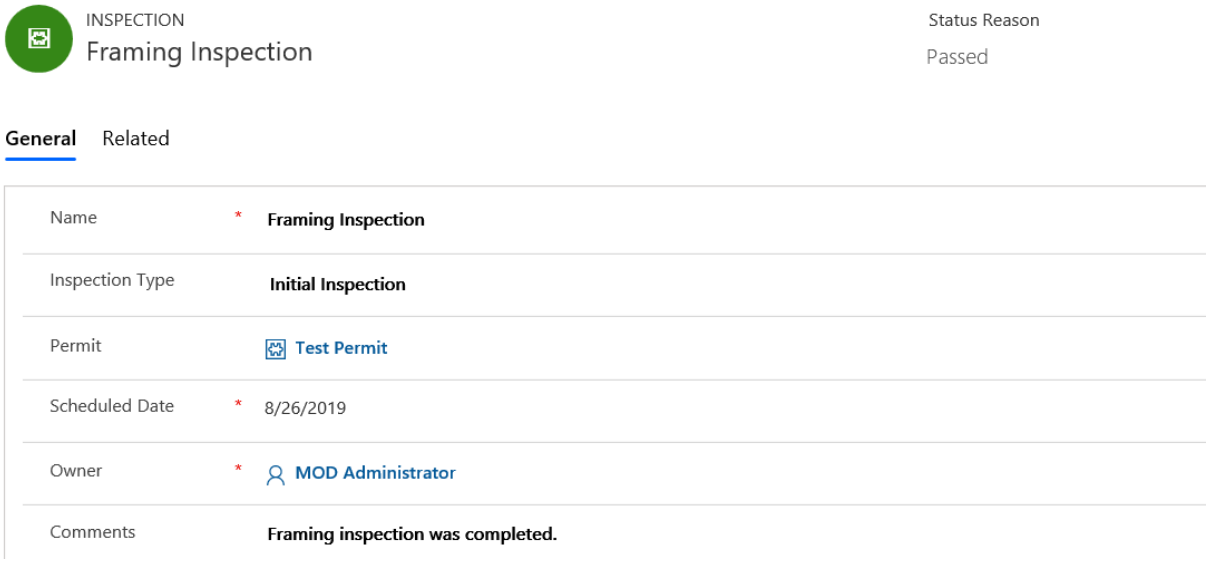
3. Confirm the inspection record was updated
   - Navigate to Power Apps maker portal
   - Select your **Dev environment.**
   - Select **Apps** and click to open the **Permit Management Application**.



   - Select **Inspections** and click to open the **Framing Inspection**.



- The **Status Reason** of the inspection should be **Passed,** and the comment should be updated to the comment you provided.



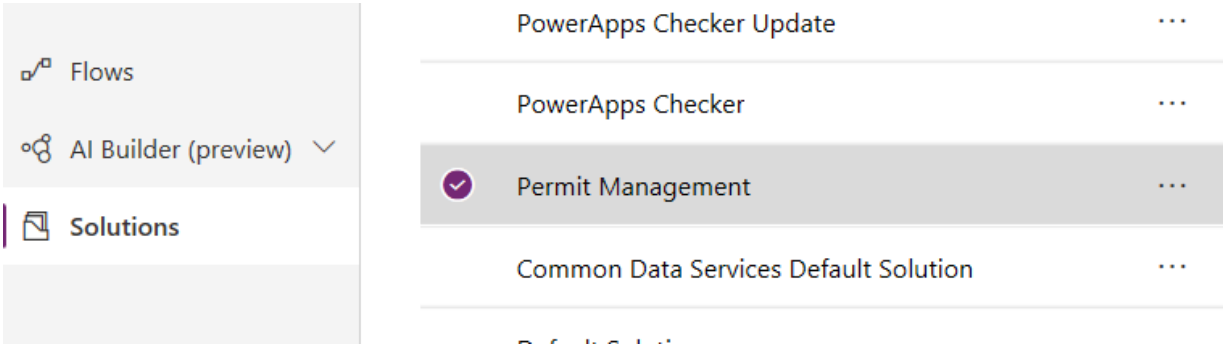
- Close the **Permit Management** application.

16 Exercise #2: Export/Import Solution

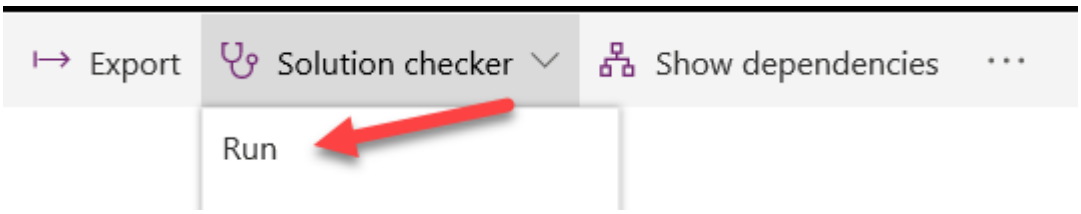
Objective: In this exercise, you will export the solution you created in the development environment and import it to the production environment.

16.1 Task #1: Export solution.

1. Select the Permit Management solution.
   - Sign in to Power Apps maker portal
   - Make sure you have your **Dev** environment selected.
   - Select **Solutions** and select the **Permit Management** solution.

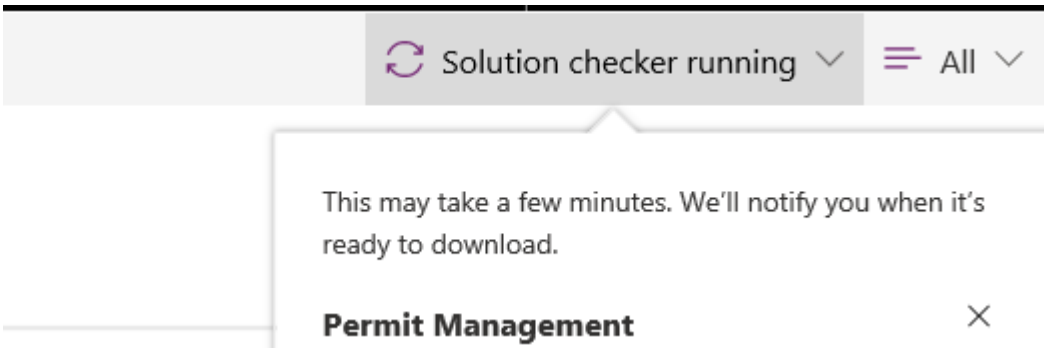


2. Run solution checker.
   - Click **Solution Checker** and select **Run**.



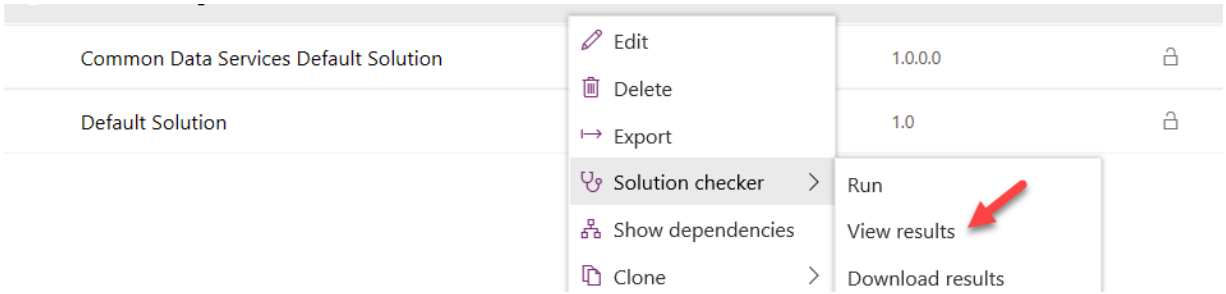
   - Wait for the run to complete.

Note: At the creation of this lab, the solution checker did not complete successfully. If you get an error with the message "Couldn't be complete" skip to step 3.

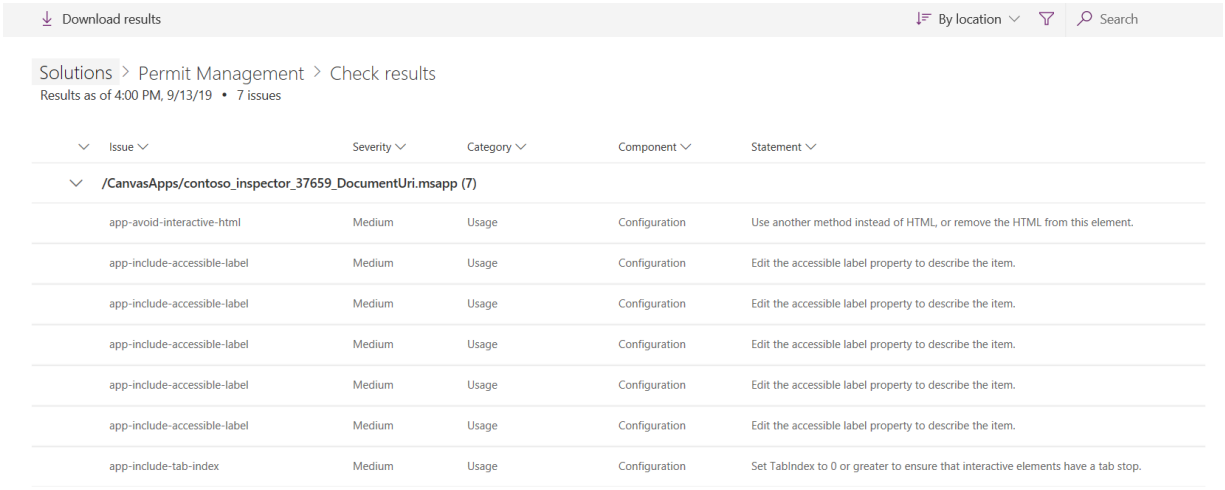


   - Click on the More **Commands** of the **Permit Management** solution.

   - Click Solution Checker and select View Results.

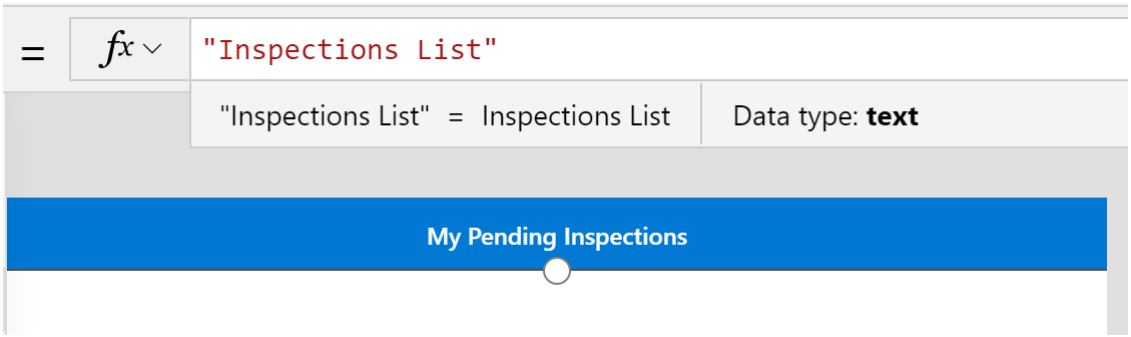


   - You will see several issues reported



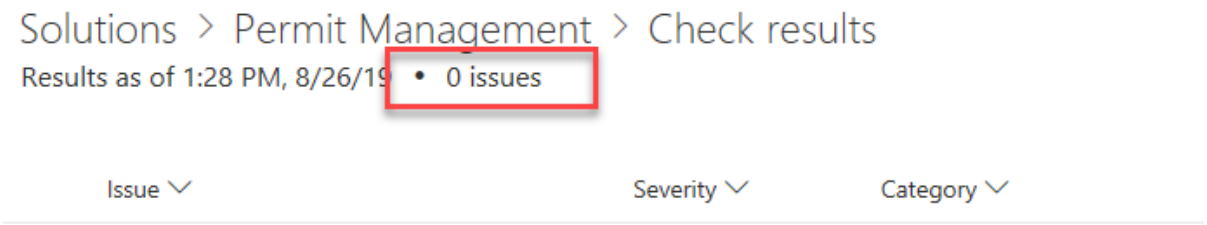
- To resolve the issues, follow these steps:
  - Select **Apps**
  - Click … next to **Inspector** app and select **Edit**
  - Click **App checked** icon on the toolbar

- Select **Recheck All**
- Expand **Missing accessible label** node
- Select an issue. This will open the screen with the control and prompt to enter **AccessibleLabel** property.
- Enter text value as appropriate

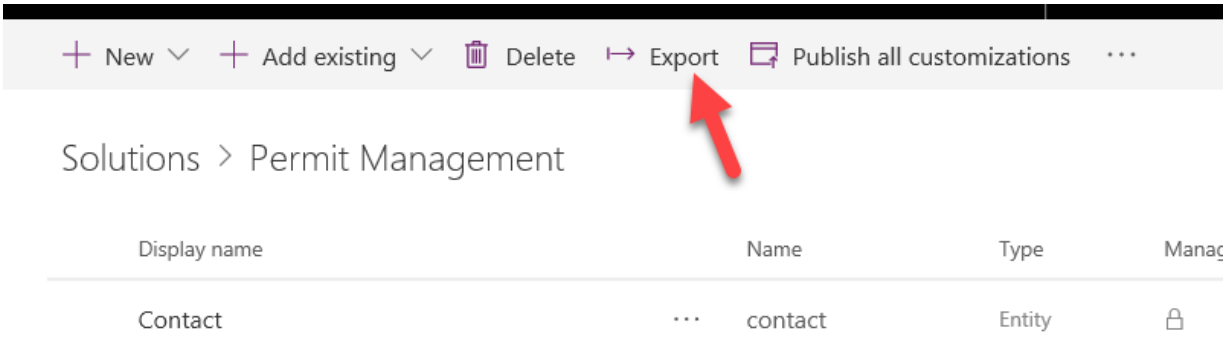


  - Repeat the process for all controls with missing accessible labels
  - Expand **Missing tab stop** node
  - Select control, enter a value for the **TabIndex**, e.g. 0
  - **Tips** node may contain the following message
    "Use another method instead of HTML, or remove the HTML from this element."
  - This message is related to Map component we imported as part of the component bundle. This component can be safely deleted as it's not used by the app.
  - Fix other app issues as appropriate.
- Click **File** and then click **Save**.
- Click **Publish**.

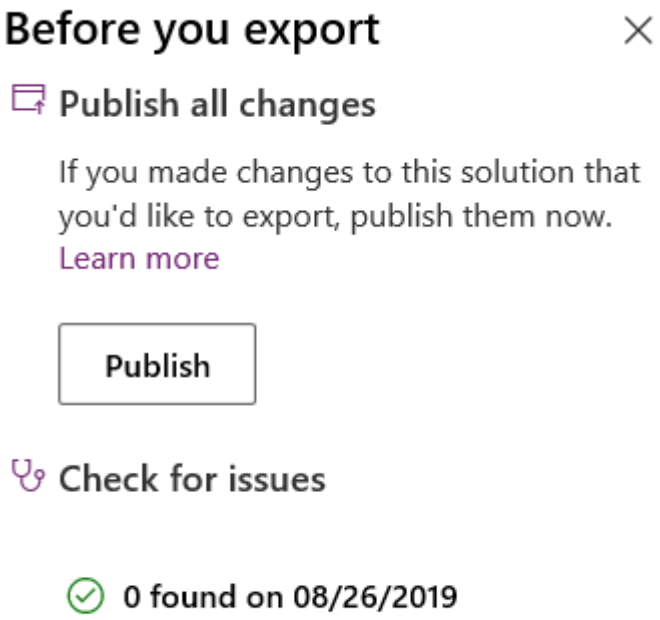
- Switch to Power Apps maker portal
- Select **Solutions** then select **Permit Management** solution
- Click **Solution checker** then select **Run** and wait for the run to complete.
- There should be zero issues.



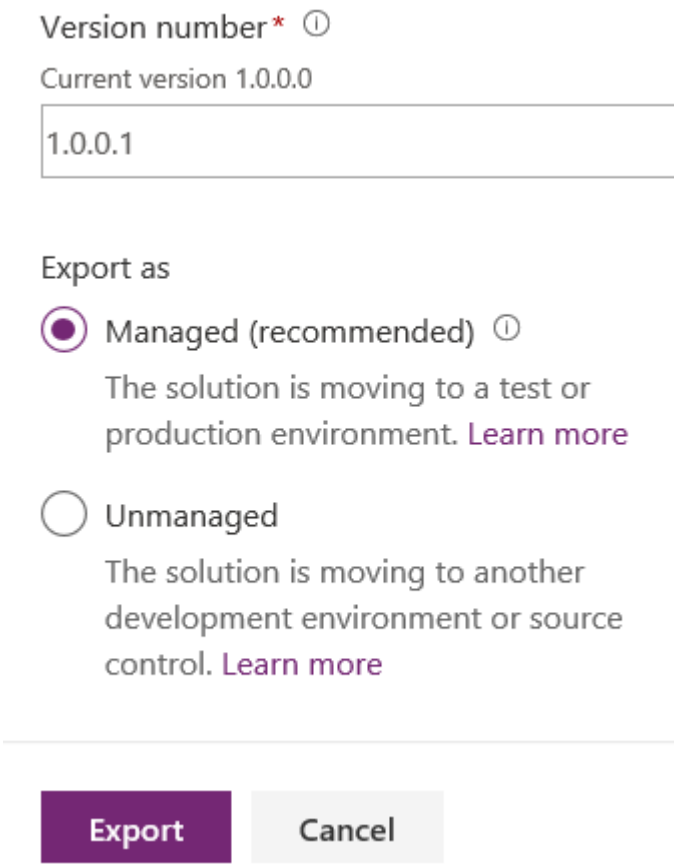
3. Export managed solution
   - Select **Solutions** and click to open the **Permit Management** solution.
   - Click **Export**.



   - Click **Publish** and wait for the publishing to complete.

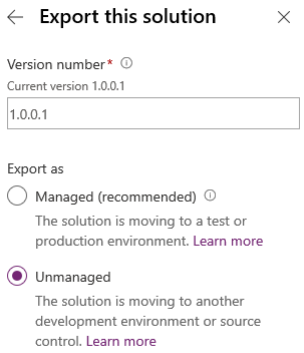


- Click **Next**.
- Select **Managed** and click **Export**.



- Save the solution on your machine.

4. Export unmanaged solution
   - Click **Export** again.
   - Click **Next**.
   - Edit the version number to match the Managed solution you just exported, select **Unmanaged** and click **Export**.



   - Save the unmanaged solution on your machine.

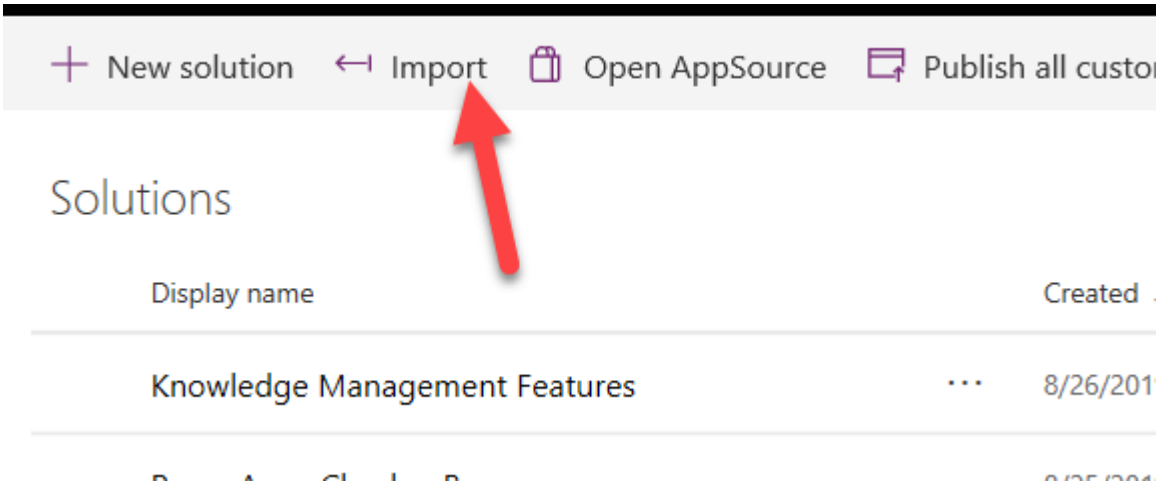
16.2 Task #2: Import solution.

1. Open the Permit Management solution.
   - Sign in to Power Apps maker portal

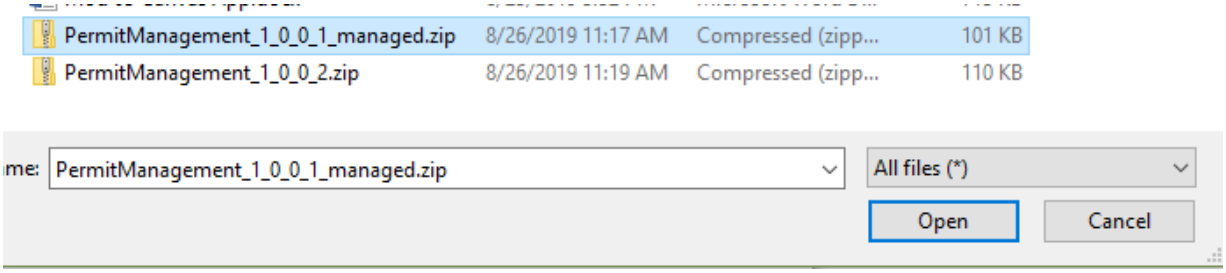
- Make sure you have your **Prod** environment selected. **Note** If you are using the community plan environment as instructed (e.g: [username]'s Environment), you may need to create a data base before you proceed to the next step.

2. Import solution

- Select **Solutions** and click **Import**.



- Click **Browse**.
- Select the **Managed** solution you exported and click **Open**.



- Click **Next**.
- Click **Import** and this will open a new window to track the import status.
- Wait for import to complete and click **Close**.
- Navigate to both the model driven and canvas apps you've created and add a few records, test the apps.

16.3 lab: title: 'Lab 04: Client Scripting'

[!NOTE] Effective November 2020:

- Common Data Service has been renamed to Microsoft Dataverse. Learn more
- Some terminology in Microsoft Dataverse has been updated. For example, *entity* is now *table* and *Column* is now *column*. Learn more

This content will be updated soon to reflect the latest terminology.

16.4 Lab 04 – Client Scripting

17 Scenario

A regional building department issues and tracks permits for new buildings and updates for remodeling of existing buildings. Throughout this course, you will build applications and automation to enable the regional building department to manage the permitting process. This will be an end-to-end solution which will help you understand the overall process flow.

In this lab, you will implement client-side logic that will use the web API to evaluate the permit type associated with the permit record and use the client scripting API to manipulate the form controls.

You will also customize the command bar to introduce a new lock permit button that will invoke a custom action to perform the lock permit logic. The server-side logic for the lock permit custom action will be implemented later in the course. Right now, you will just add the button and the logic to invoke the action.

17.1 High-level lab steps

As part of building the client-side logic, you will complete the following:

- Setup a folder to contain your client script
- Upload and register the client script on the form
- Build logic to use the web API to retrieve the permit type record associated with the permit
- Build logic based on the permit type settings to hide and show the inspections tab on the form
- Build logic to set Columns as required/not required based on the permit type settings
- Use a community tool, Ribbon Workbench, to modify the command bar
- Build logic to invoke the lock permit custom action when the command bar button is clicked

17.2 Things to consider before you begin

- Are there alternative designs that would be viable and not require code?
- Remember to continue working in your DEVELOPMENT environment. We will move everything to production soon.

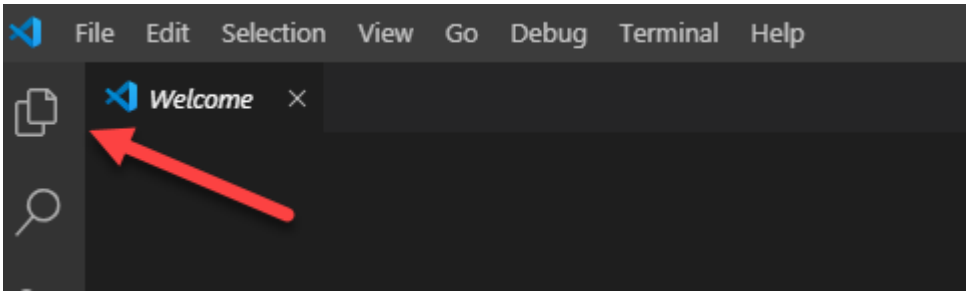
18 Exercise #1: Prepare and Load Resources

Objective: In this exercise, you will create, organize, and load your JavaScript web resources.

18.1 Task #1: Use Visual Studio Code to Create Resources

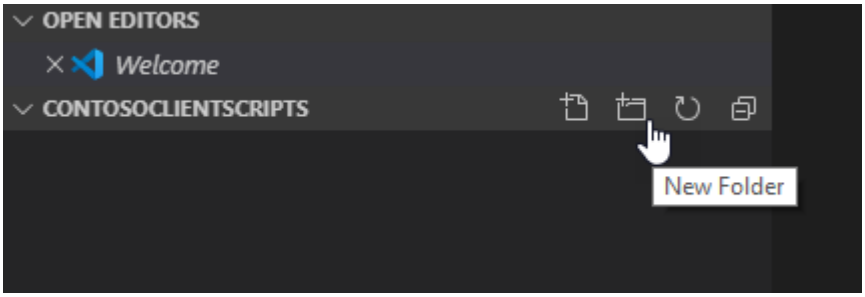
In this task, you will set up a folder to contain the JavaScript web resource files in this course.

1. If you do not already have Visual Studio Code, download it from here Visual Studio Code and install it.
2. Start **Visual Studio Code**.
3. Create resources
   - Select Explorer from left menu or press Ctrl + Shift + E.

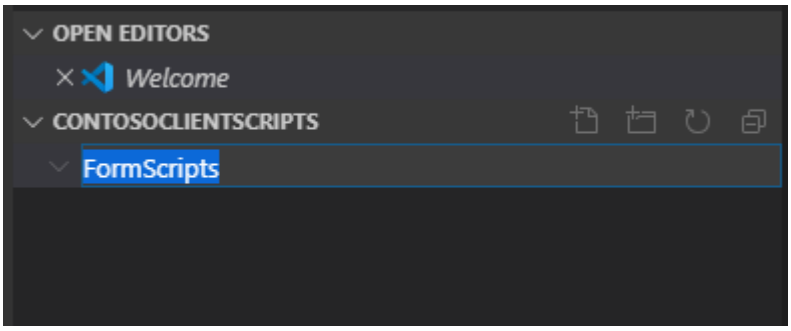


   - Click **Open Folder**.
   - Create a new folder and name it **ContosoClientScripts**.
     Note: This is the name and structure used for this lab, the platform does not require a specific structure or content organization. Many projects check these assets into a source control system to keep track of all the changes over the life of the client script.
   - Select the new folder you just created and click **Select Folder**.

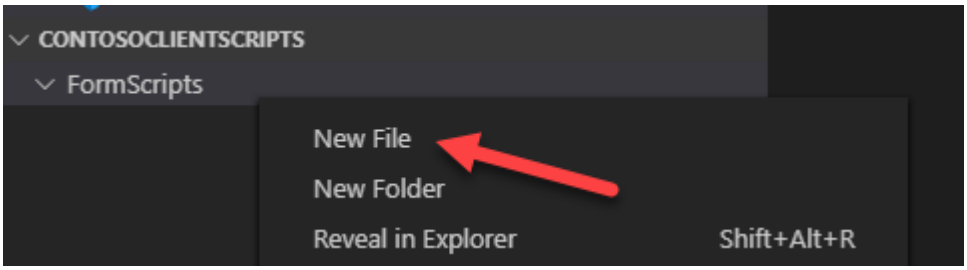
4. Create **Form Scripts** folder
   - Hover over the folder and click **New Folder**.



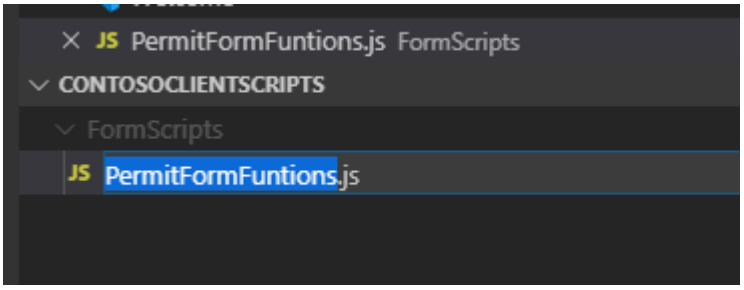
   - Name the new folder as **FormScripts** and click **Enter**.



5. Create the **Permit Form Functions** file
   - Right click on the **FormScripts** folder and select **New File**.



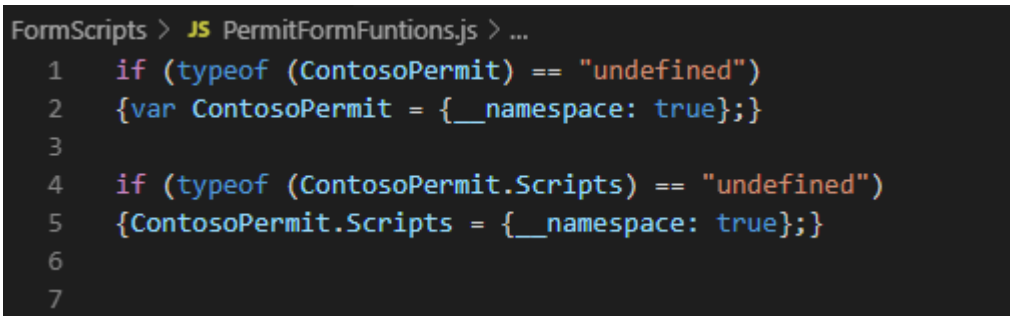
   - Name the new file as **PermitFormFuntions.js** and click **Enter**.



   - Add the below mentioned namespaces to the newly created **PermitFormFunctions** file.

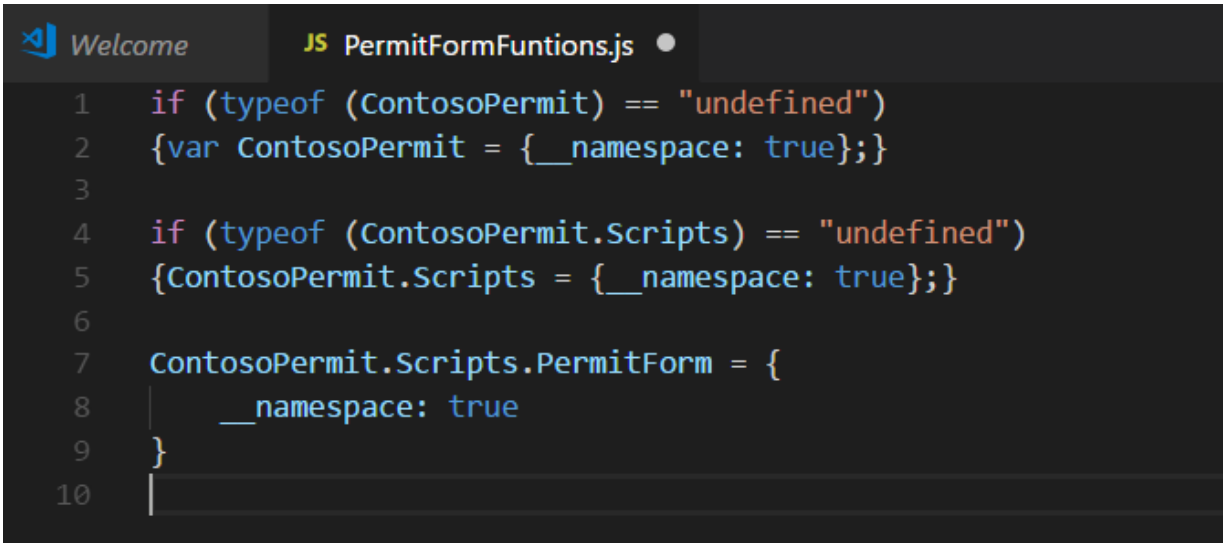
```
if (typeof (ContosoPermit) == "undefined")
{var ContosoPermit = {__namespace: true};}
```

```
if (typeof (ContosoPermit.Scripts) == "undefined")
{ContosoPermit.Scripts = {__namespace: true};}
```



- Add the function mentioned below after adding the namespaces.

```
ContosoPermit.Scripts.PermitForm = {
__namespace: true
}
```



18.2 Task #2: Add Event Handlers

In this task, you will create functions for the logic that you will be implementing. This will allow you to register the event handlers in the next tasks for calling these functions and performing few basic tests in the upcoming tasks.

1. Add a function to OnLoad event
   - Add the function mentioned below to the **PermitFormFuntions** file inside the function created in Step 1(d).

```
handleOnLoad: function (executionContext) {
console.log('on load - permit form');
},
```

```
6
7   ContosoPermit.Scripts.PermitForm = {
8
9       handleOnLoad: function (executionContext) {
10          console.log('on load - permit form');
11      },
12
13      __namespace: true
14  }
```

2. Add a function to OnChange permit type event
   - Add the function mentioned below to the **PermitFormFuntions** file inside the function created in Step 1(d). Once this is done, click **File** and **Save All**.

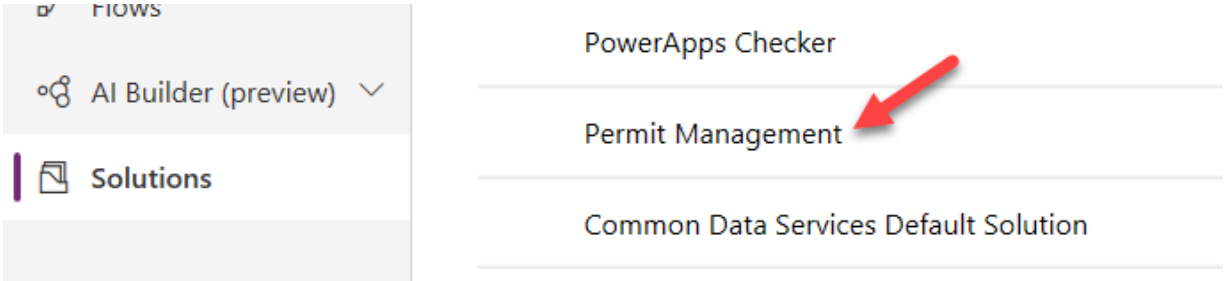
```
handleOnChangePermitType: function (executionContext) {

console.log('on change - permit type');

},
```

```
7   ContosoPermit.Scripts.PermitForm = {
8
9       handleOnLoad: function (executionContext) {
10          console.log('on load - permit form');
11      },
12
13      handleOnChangePermitType: function (executionContext) {
14          console.log('on change - permit type');
15      },
16
17      __namespace: true
18  }
```

18.3 Task #3: Load Web Resources

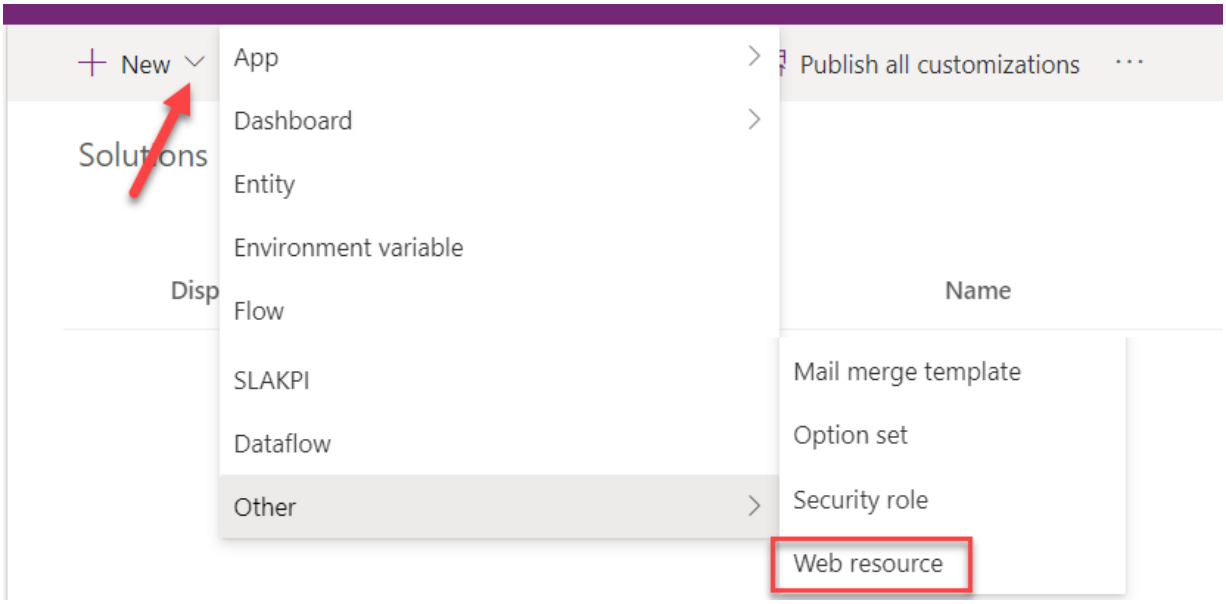
In this task, you will upload the JavaScript files as web resources. Here, you will also edit the Permit Table main form and associate the new web resource with its form. Finally, you will register your functions to be called on specific form events.

1. Open the Permit Management solution
   - Sign in to Power Apps maker portal.
   - Select your **Dev** environment.
   - Select **Solutions**.
   - Click to open the **Permit Management** solution.

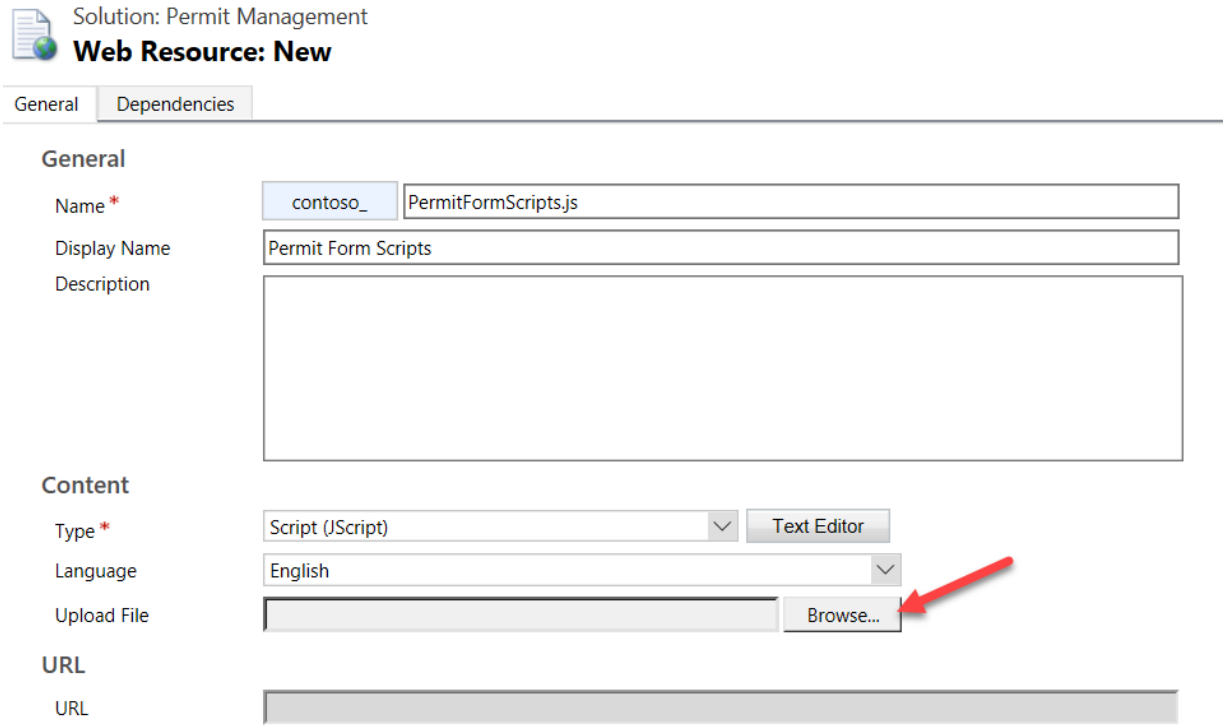


2. Add web resource to the solution

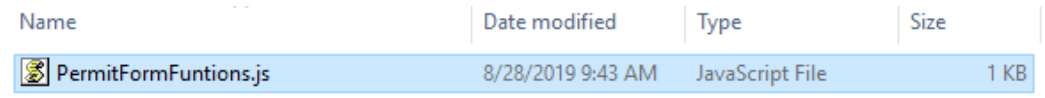
- Click + **New**.
- Click **Other** and select **Web Resources**.



- Enter **PermitFormScripts.js** for **Name**.
- Enter **Permit Form Scripts** for **Display Name**.
- Select **Script (Jscript)** for **Type**.
- Select **English** for **Language**.
- Click **Browse**.

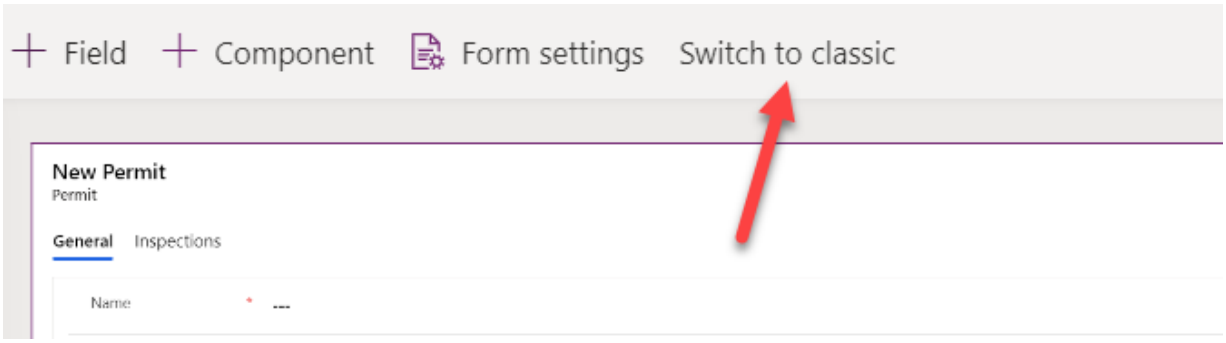


- Select the **PermitFormFunctions.js** file and click **Open**.

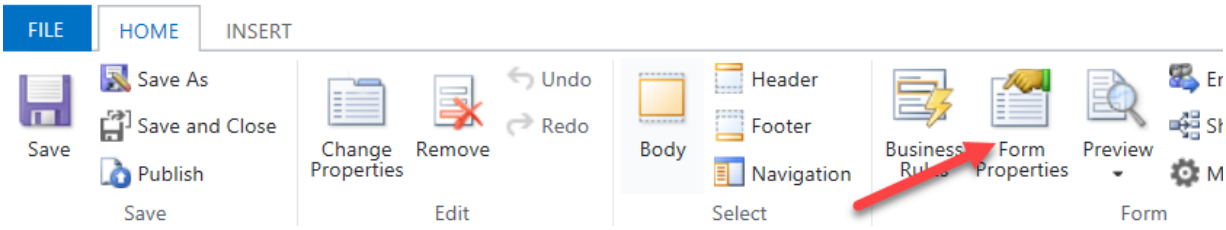


- Click **Save** and wait until the changes are saved.
- Click **Publish** and wait for the publishing to complete.
- Close the web resource editor Window.
- Click **Done**.

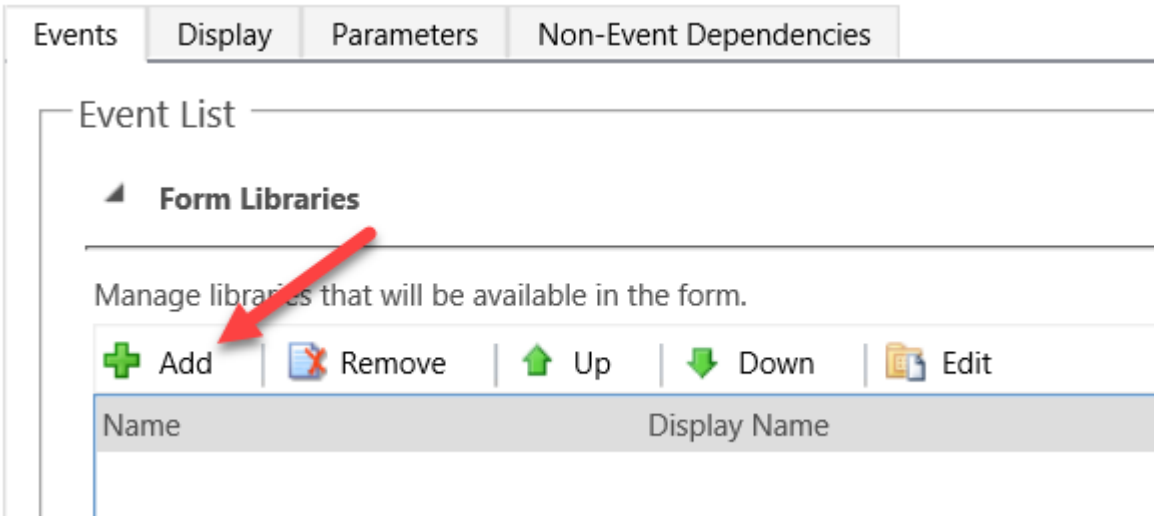
3. Open the Permit main form.
   - Make sure you are still in the solution.
   - Click to open the **Permit** Table.
   - Select the **Forms** tab and click to open the **Main** form.
4. Switch to Classic.
   Note: Currently, you must switch to the classic solution explorer to complete the hookup of the client script to the form. In the future, this will be moved to make.powerapps.com and will not require switching to the classic tools.
   - Click on the **Switch to Classic**. This will open the classic UI in a new browser window.



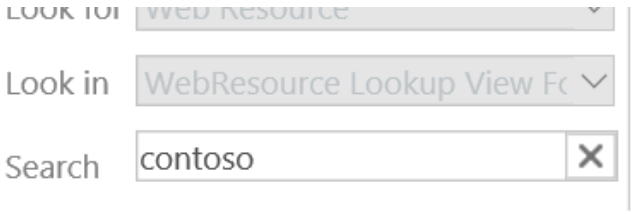
5. Add the script to the permit form
   - Click **Form Properties**.

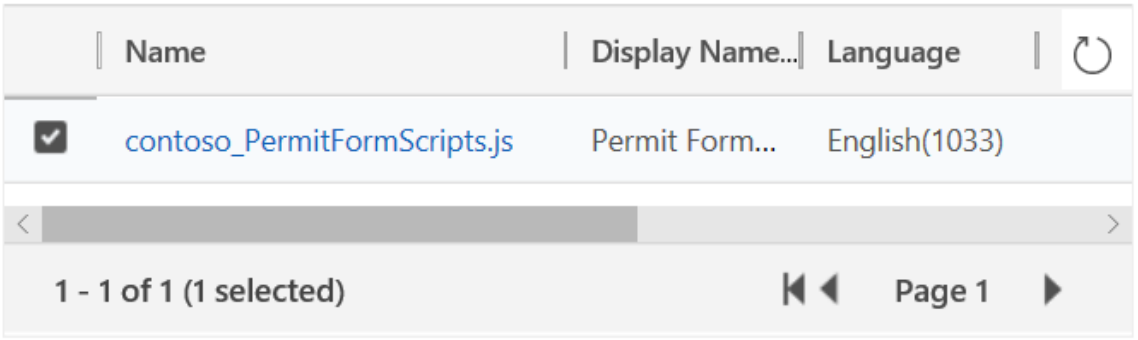


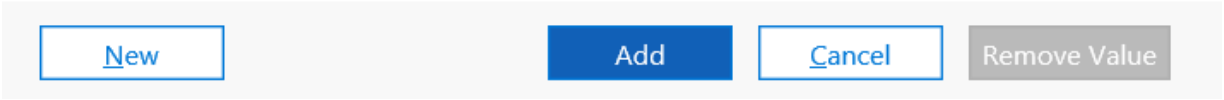
   - Click **Add Library**.



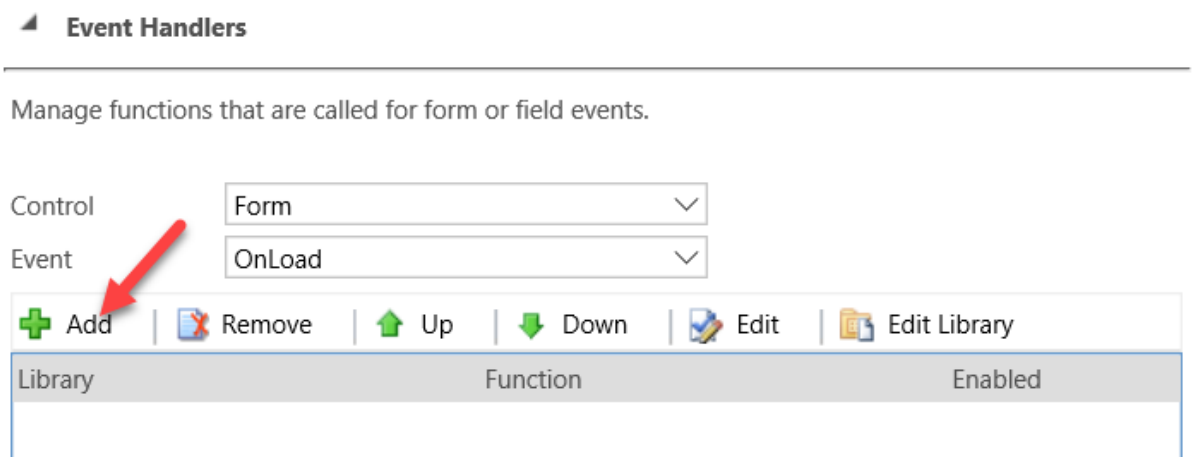
- Search for **contoso**, select **PermitFormScripts** entry with Display Name as **PermitFormScripts** and click **Add**.



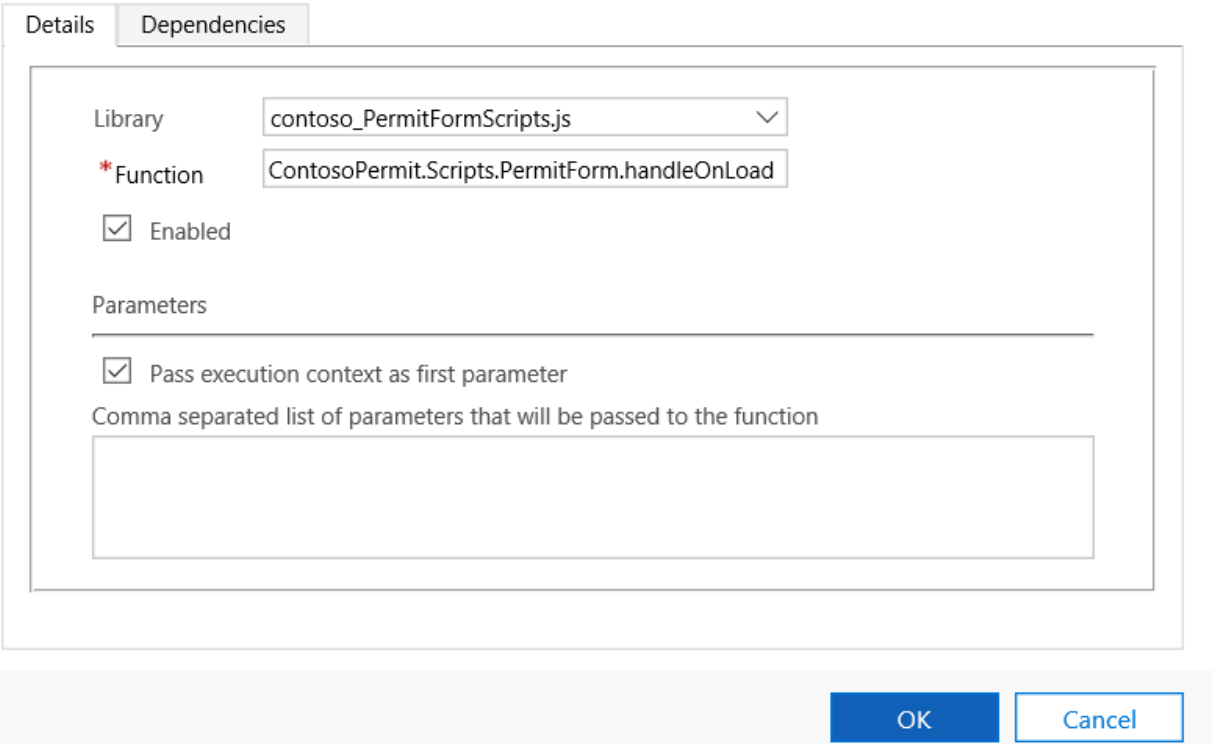




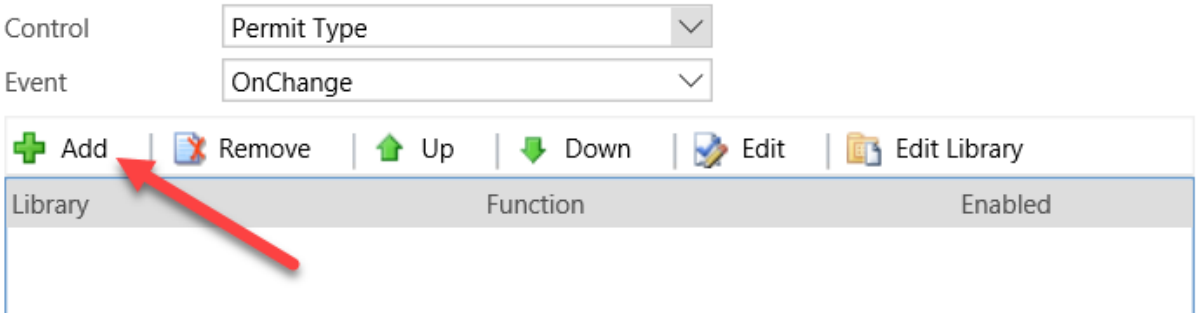
6. Add OnLoad event handler.
   - Go to the **Event Handlers** section.
   - Select **Form** from **Control Dropdown**.
   - Select **OnLoad** from **Event Dropdown**.
   - Click **Add**.



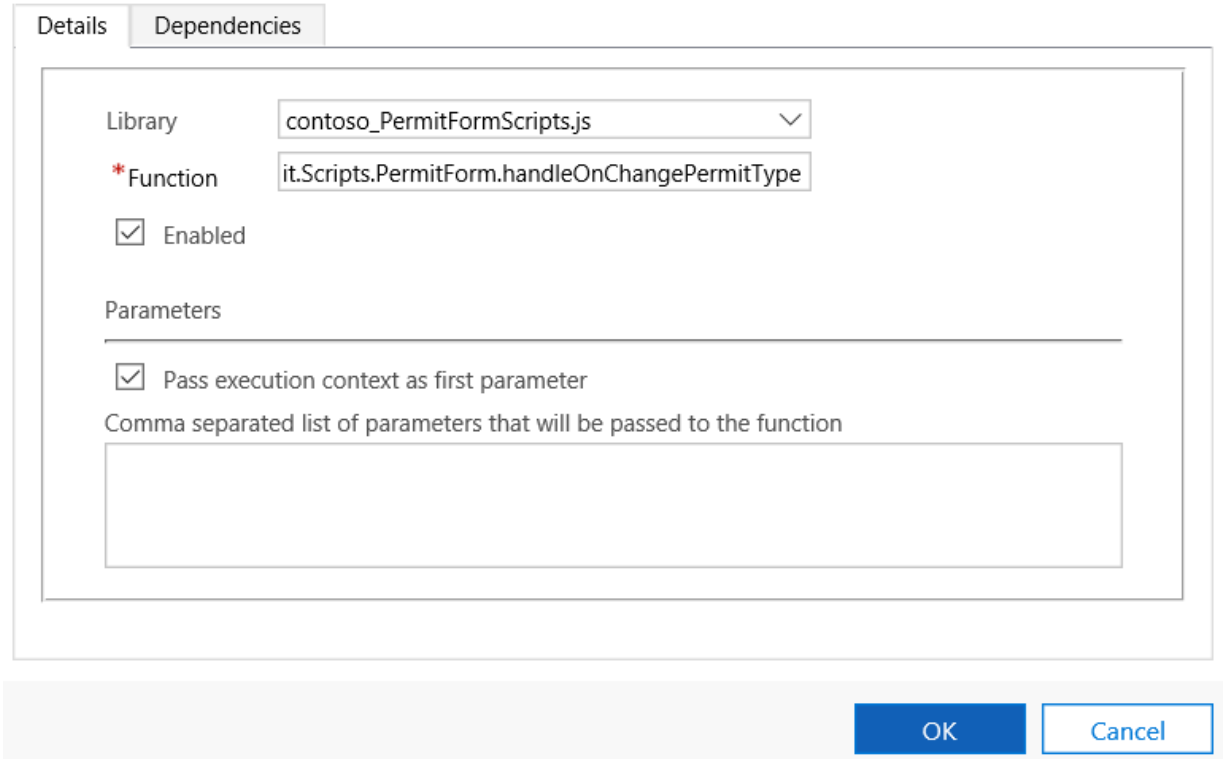
- Select **Contoso_**PermitFormScripts.js** in the dropdown for **Library**.
- Enter **ContosoPermit.Scripts.PermitForm.handleOnLoad** in the textbox for **Function**.
- Check the **Pass Execution Context** checkbox.
- Click **OK**.



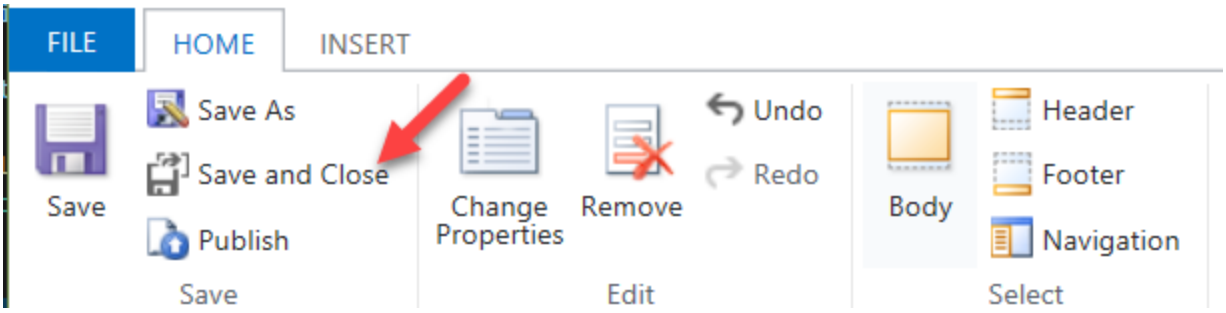
7. Add Permit Type OnChange event handler.
   - Go to the **Event Handlers** section.
   - Select **Permit Type** from the dropdown for **Control**.
   - Select **OnChange** from the dropdown for **Event**.
   - Click **Add**.



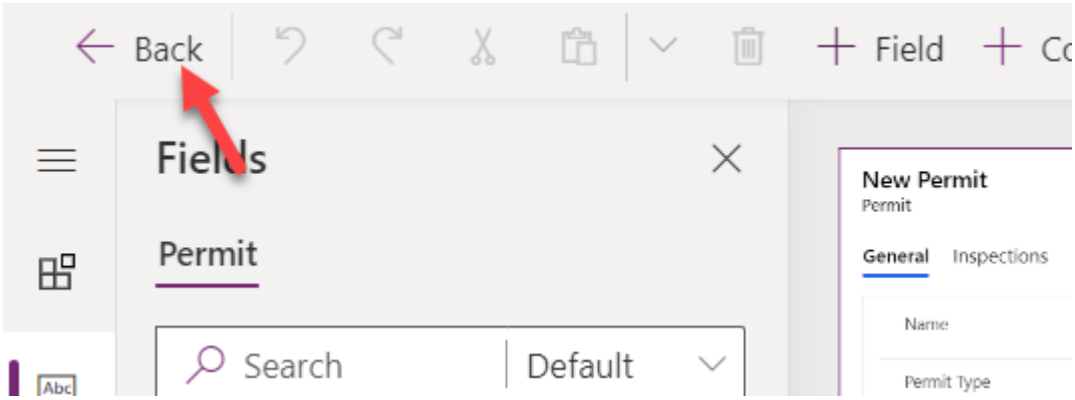
- Select **Contoso_PermitFormScripts.js** for **Library**.
- Enter **ContosoPermit.Scripts.PermitForm.handleOnChangePermitType** in the textbox for **Function**.
- Check the **Pass Execution Context** checkbox.
- Click **OK**.



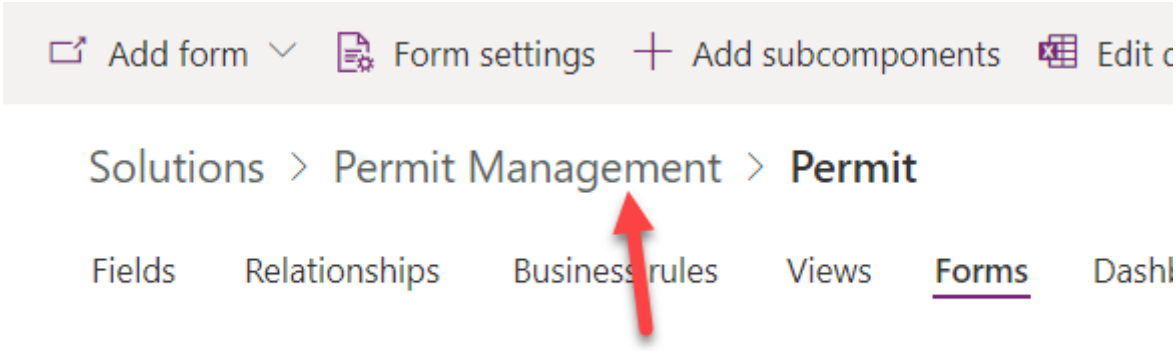
8. Save and publish your changes
   - Click **OK** to close the **Form Properties** window.
   - Click **Save and Close**.



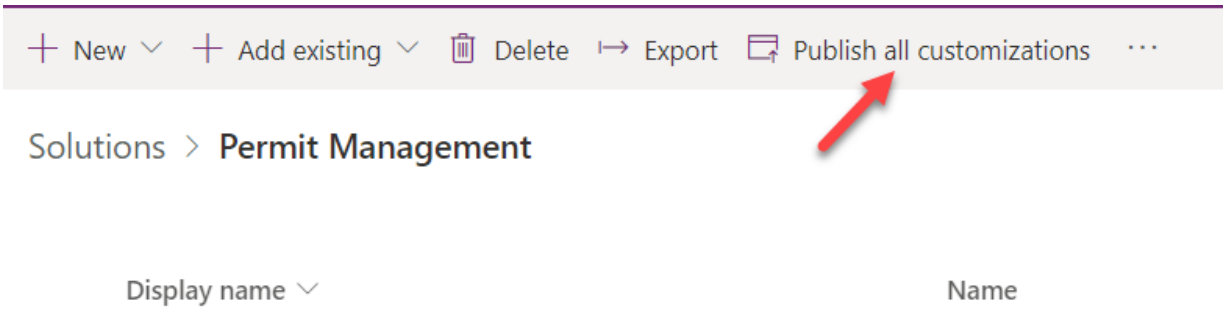
   - Click on the **<- Back** button.



- Go back to the solution by clicking on the solution name.



- Click **Publish All Customizations** and wait for the publishing to complete.

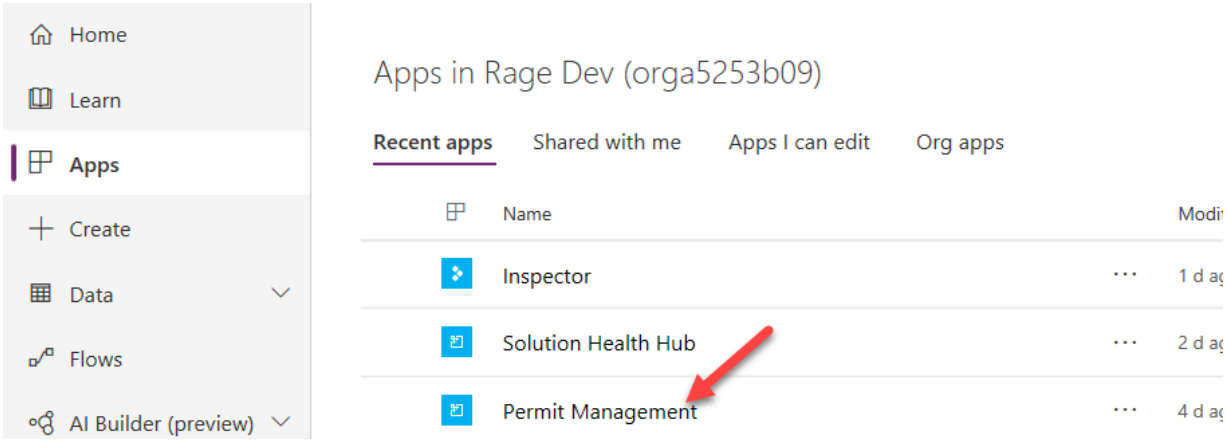


DO NOT navigate away from this page

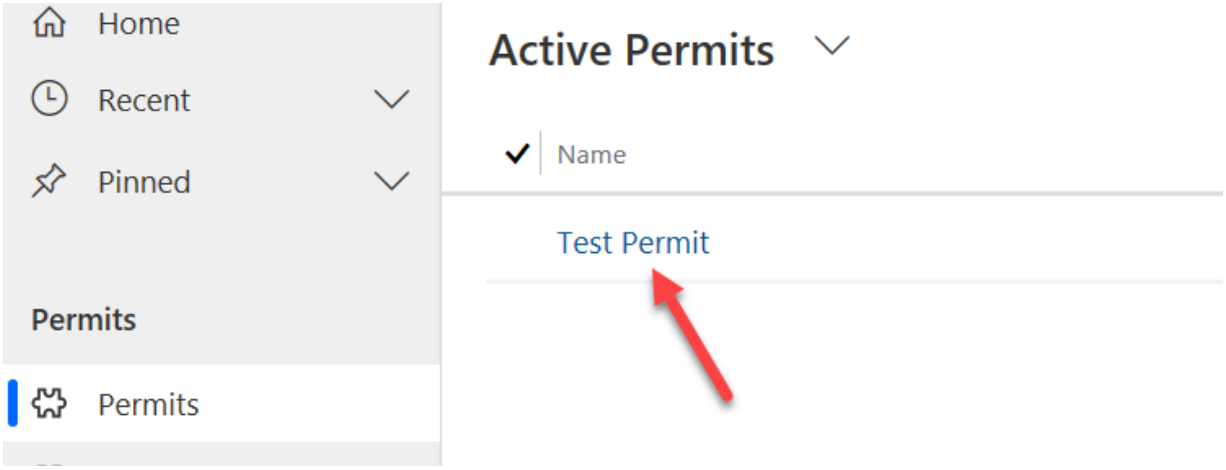
18.4 Task #4: Test Event Handlers

In this task, you will be doing a test to ensure that you have correctly hooked up your functions to the permit form events.

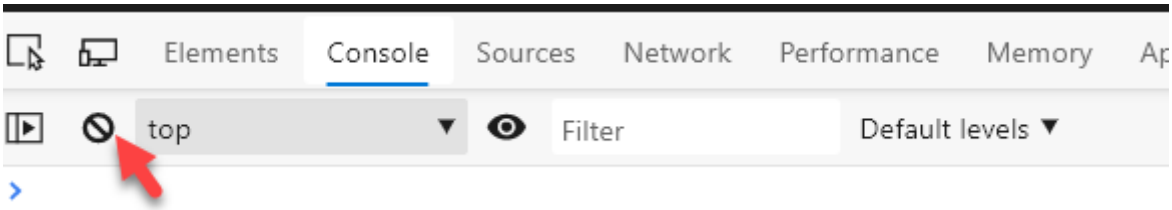
1. Start the Permit Management application
   - Select **Apps**.
   - Click to start the **Permit Management** application and this will open the app in New Window.



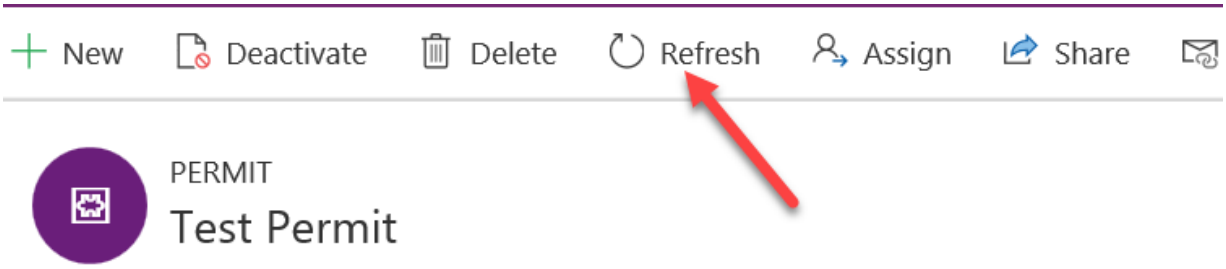
2. Open a Permit record
   - Select **Permits** from the Site Map.
   - Click to open a permit record.



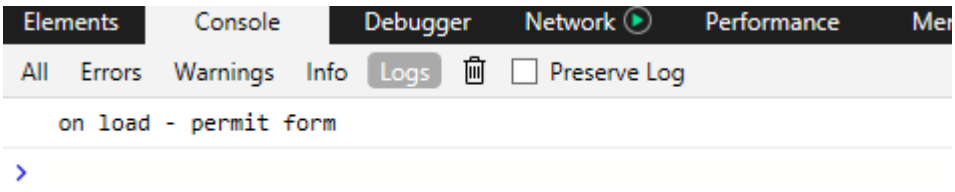
3. Open Edge Dev Tools
   - Press **F12** or right click and select **Inspect**.
   - Select the **Console** from top menu and clear console.



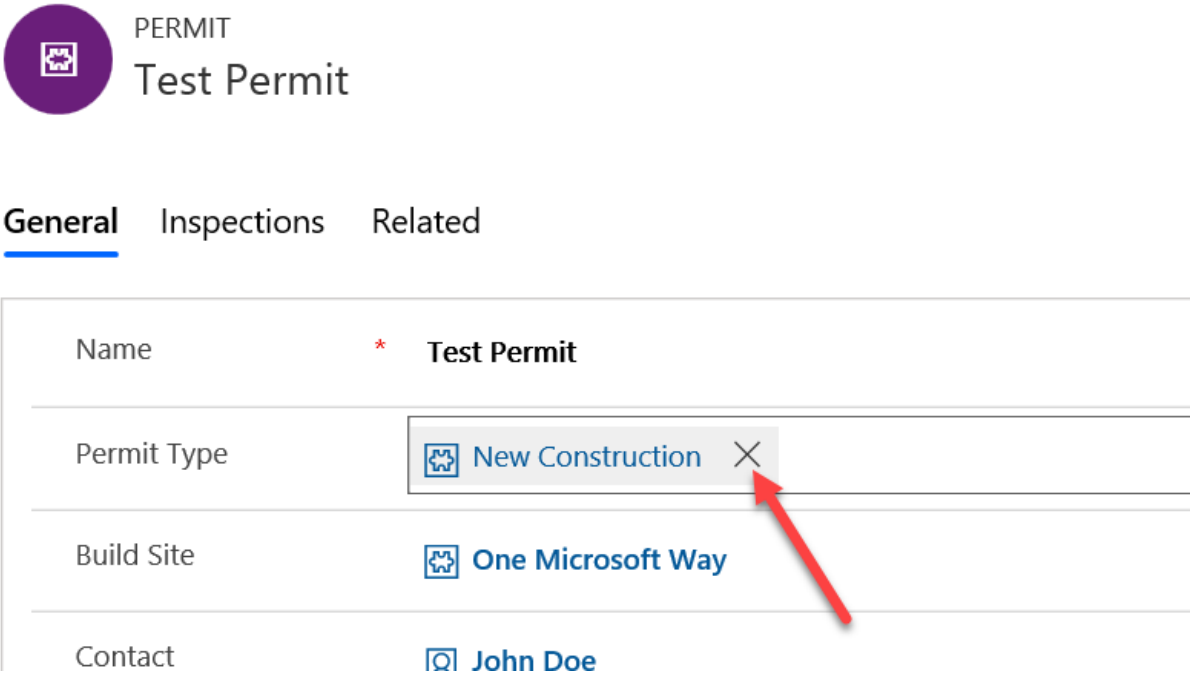
4. Refresh and confirm the OnLoad event handler function runs
   - Go to the **Permit** record and click **Refresh**.



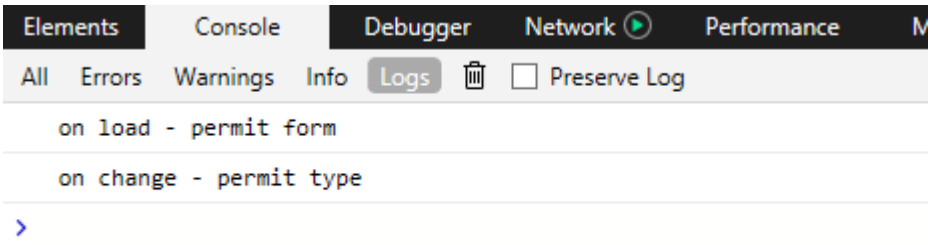
   - Go to the **Dev Tools** and you should now be able to see the **on load – permit form** message.



5. Remove Permit Type and confirm the OnChange Permit Type event handler function runs
   - Go to the **Permit** record and remove the **Permit Type**.



   - Go to the **Dev Tools** and you should now be able to see the **on change – permit type** message.



   - Close the **Dev Tools**.

19 Exercise #2: Show and Hide Tabs

Objective: In this exercise, you will create a script that will show and hide the inspections tab based on the permit type Table's "required inspections" Column value.

19.1 Task #1: Create Function

1. Create a function that will run when the Permit form loads and when the Permit Type value changes
   - Go back to **Visual Studio Code**.
   - Add the function mentioned below to **PermitFormFuntions** inside the PermitForm function.

```
_handlePermitTypeSettings: function (executionContext) {
```

```
},
```

```
        console.log('on change - permit type');
    },

    _handlePermitTypeSettings: function (executionContext) {

    },
__namespace: true
}
```

2. Get form context from the execution context
   - Add the script mentioned below inside _**handlePermitTypeSettings** function.

     ```
     var formContext = executionContext.getFormContext();
     ```

3. Get the Permit Type value from the form.
   - Add the script mentioned below inside the _**handlePermitTypeSettings** function. contoso_permittype is the logical name of the Permit Type Column. You can verify this in the Table metadata.

     ```
     var permitType = formContext.getAttribute("contoso_permittype").getValue();
     ```

4. Check if the Permit Type has value.
   - Add the script mentioned below inside the _**handlePermitTypeSettings** function.

     ```
     if (permitType == null) {

     } else {

     }
     ```

```
    _handlePermitTypeSettings: function (executionContext) {
        var formContext = executionContext.getFormContext();
        var permitType = formContext.getAttribute("contoso_permittype").getValue();
        if (permitType == null) {

        } else {

        }

    },
```

5. Hide the Inspections tab and return if Permit type is null.
   - Add the script mentioned below inside the _**handlePermitTypeSettings** function. inspectionsTab is the name of the Inspections tab (This is configured while configuration of the Model Driven App in Lab1 Module 2).

     ```
     formContext.ui.tabs.get("inspectionsTab").setVisible(false);
     return;
     ```

```
},
_handlePermitTypeSettings: function (executionContext) {
    var formContext = executionContext.getFormContext();
    var permitType = formContext.getAttribute("contoso_permittype").getValue();
    if (permitType == null) {
        formContext.ui.tabs.get("inspectionsTab").setVisible(false);
        return;

    } else {

    }

},


  namespace: true
```

19.2 Task #2: Get Inspection Type Record

In this task, you will use the web API to retrieve the permit type lookup record associated with the current permit record that is currently displayed in the form.

1. Get the Permit Type ID

   - Add the script mentioned below in the else statement of the **_handlePermitTypeSettings** function.

     ```
     var permitTypeID = permitType[0].id;
     ```

2. Retrieve the Permit Type record and show alert if there are errors

   - Add the script mentioned below in the else statement of the **_handlePermitTypeSettings** function. contoso_pertmittype is the logical name of the Permit Type Table.

     ```
     Xrm.WebApi.retrieveRecord("contoso_permittype", permitTypeID).then(function (result) {
     },

     function (error) { alert('Error:' + error.message) });
     ```

```
        console.log( on change - permit type );
    },
    _handlePermitTypeSettings: function (executionContext) {
        var formContext = executionContext.getFormContext();
        var permitType = formContext.getAttribute("contoso_permittype").getValue();
        if (permitType == null) {
            formContext.ui.tabs.get("inspectionsTab").setVisible(false);
            return;

        } else {
            var permitTypeID = permitType[0].id;
            Xrm.WebApi.retrieveRecord("contoso_permittype", permitTypeID).then(function (result) {

            },
            function (error) { alert('Error:' + error.message) });

        }

    },
```

3. Check if "**Require Inspections**" Column value is true

   - Add the script mentioned below in the **retrieveRecord** function call. contoso_requireinspections is the logical name of the Require Inspections Column of the Permit Type Table.

     ```
     if (result.contoso_requireinspections) {
     ```

```
} else {

}
```

4. Make the Inspections tab visible if Require Inspections is true
   - Add the script mentioned below in the if statement of the **retrieveRecord** call.

     ```
     formContext.ui.tabs.get("inspectionsTab").setVisible(true);
     ```

5. Hide the Inspections tab if Require Inspections is not true
   - Add the script mentioned below in the else statement of the **retrieveRecord** call.

     ```
     formContext.ui.tabs.get("inspectionsTab").setVisible(false);
     ```

```
_handlePermitTypeSettings: function (executionContext) {
    var formContext = executionContext.getFormContext();
    var permitType = formContext.getAttribute("contoso_permittype").getValue();
    if (permitType == null) {
        formContext.ui.tabs.get("inspectionsTab").setVisible(false);
        return;

    } else {
        var permitTypeID = permitType[0].id;
        Xrm.WebApi.retrieveRecord("contoso_permittype", permitTypeID).then(function (result) {
            if (result.contoso_requireinspections) {
                formContext.ui.tabs.get("inspectionsTab").setVisible(true);
            } else {
                formContext.ui.tabs.get("inspectionsTab").setVisible(false);
            }

        },
        function (error) { alert('Error:' + error.message) });

    }

},
```

6. Call the _handlePermitTypeSettings function from the handleOnLoad function.
   - Go to the **handleOnLoad** function and add the script mentioned below.

     ```
     ContosoPermit.Scripts.PermitForm._handlePermitTypeSettings(executionContext);
     ```

7. Call the _handlePermitTypeSettings function from the handleOnChangePermitType function.
   - Go to the **handleOnChangePertmitType** function and add the script mentioned below.

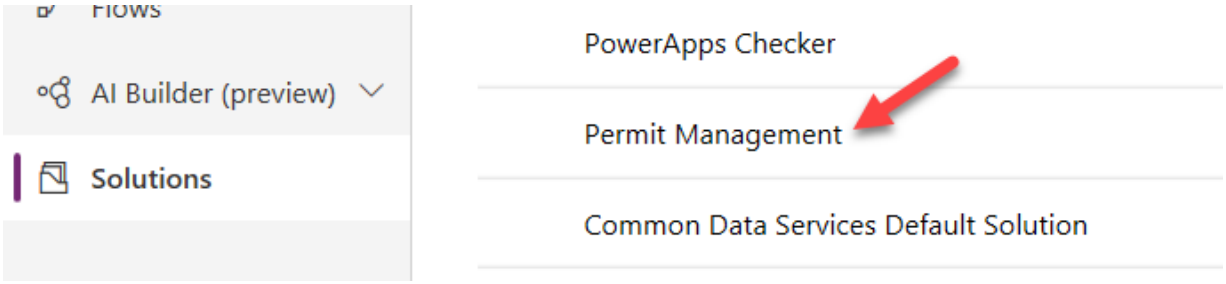
     ```
     ContosoPermit.Scripts.PermitForm._handlePermitTypeSettings(executionContext);
     ```

```
ContosoPermit.Scripts.PermitForm = {

    handleOnLoad: function (executionContext) {
        ContosoPermit.Scripts.PermitForm._handlePermitTypeSettings(executionContext);
        console.log('on load - permit form');
    },

    handleOnChangePermitType: function (executionContext) {
        ContosoPermit.Scripts.PermitForm._handlePermitTypeSettings(executionContext);
        console.log('on change - permit type');
    },

    handlePermitTypeSettings: function (executionContext) {
```

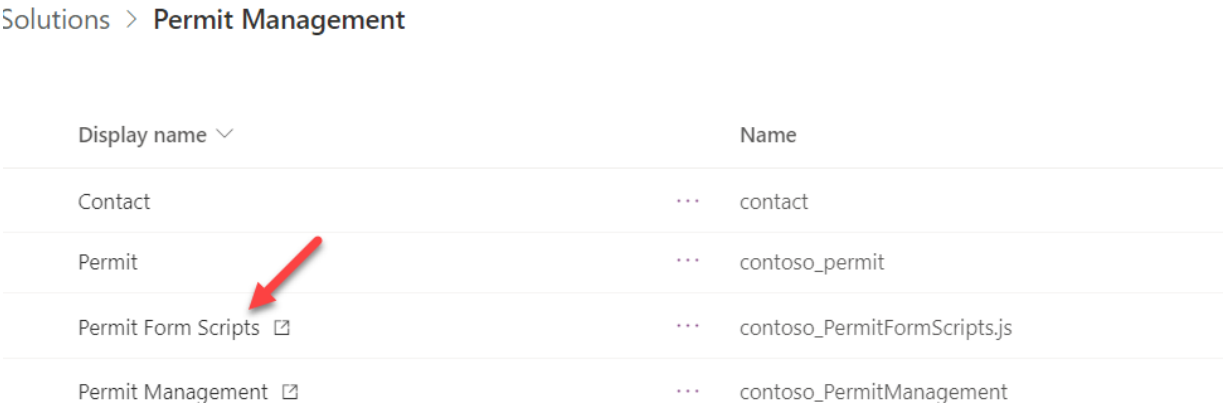
Click File and **Save All**.

19.3 Task #3: Load Updated Script

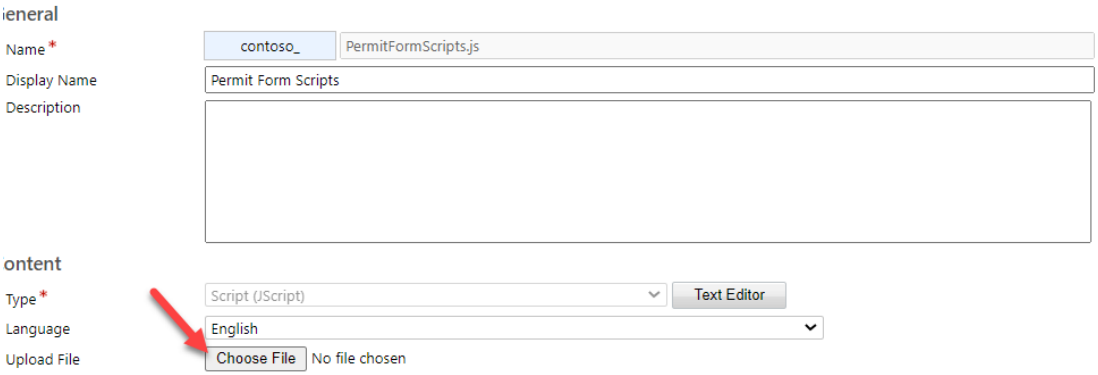
1. Open the Permit Form Script web resource.
   - Navigate to Power Apps maker portal.
   - Select your **Dev** environment.
   - Select **Solutions**.
   - Click to open the **Permit Management** solution.



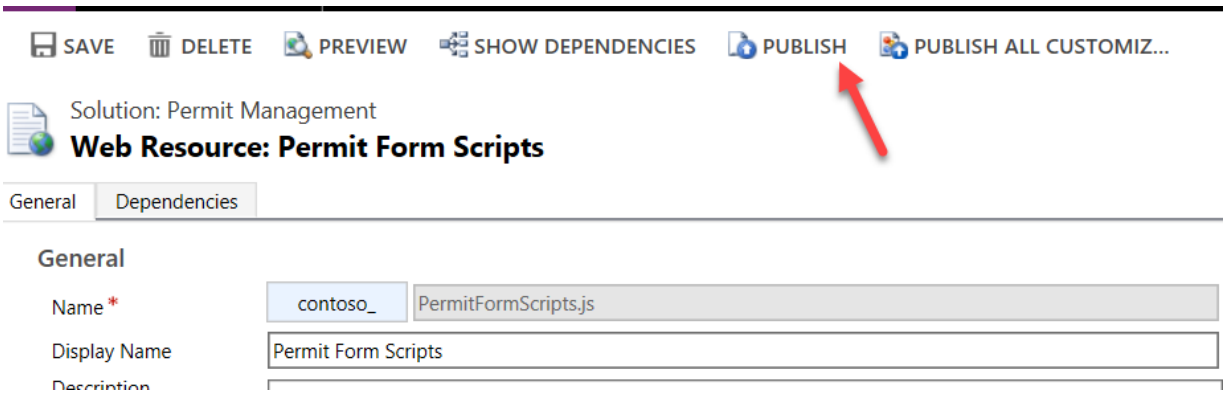
   - Locate and click to open the **Permit Form Script** web resource.



2. Load the updated version of permitFormFuntion.jsPermitFormFuntion.js
   - Click **Choose file**.



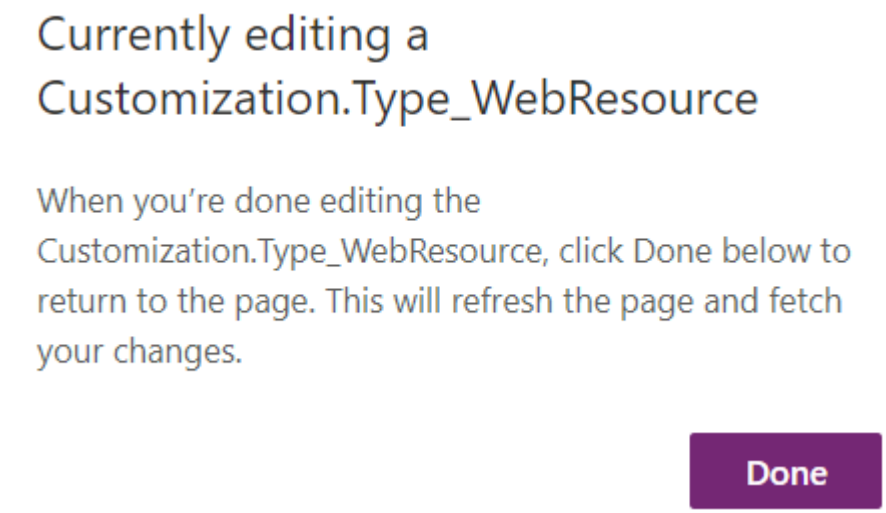
   - Select **PermitFormFunctions.js** and click **Open**.
3. Save and Publish your changes
   - Click **Save** and wait until the changes are saved.
   - Click **Publish** and wait for the publishing to complete.



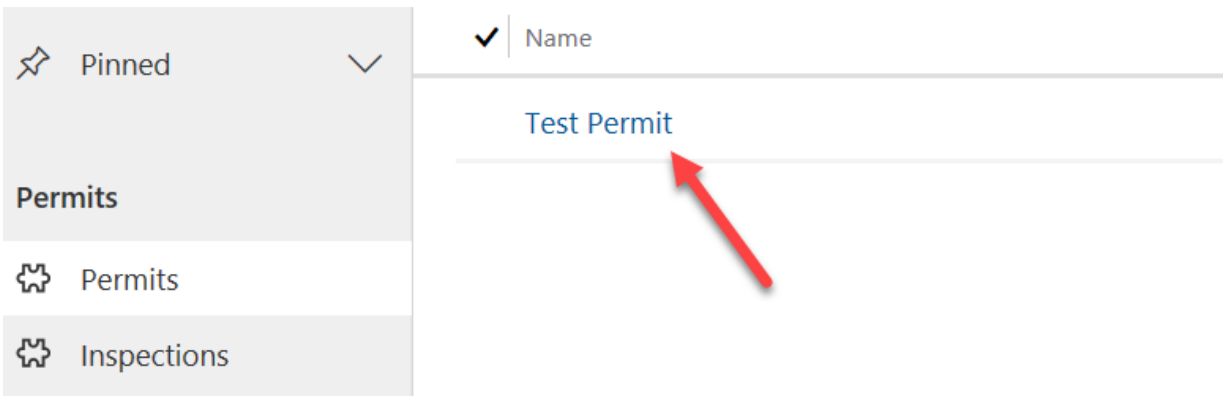
DO NOT close this window. You will need to come back to this window in the next exercise.

19.4 Task #4: Test Your Changes

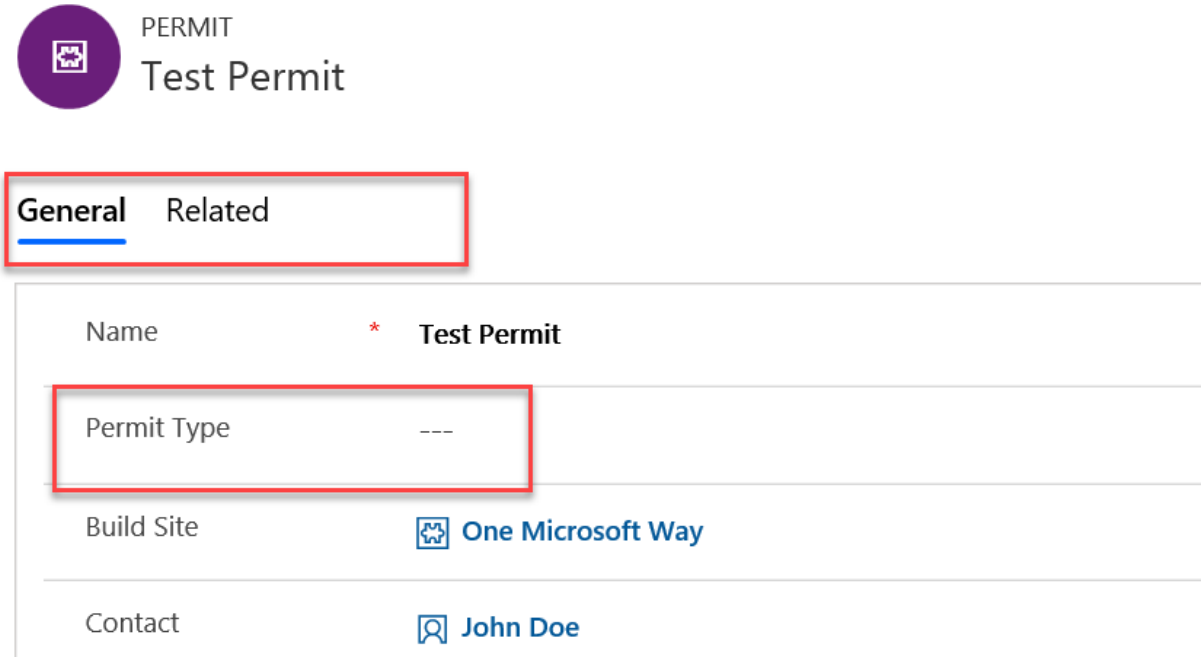
1. Start the Permit Management application
   - Select the browser tab or window for the Power Apps maker portal and click **Done** on the popup.



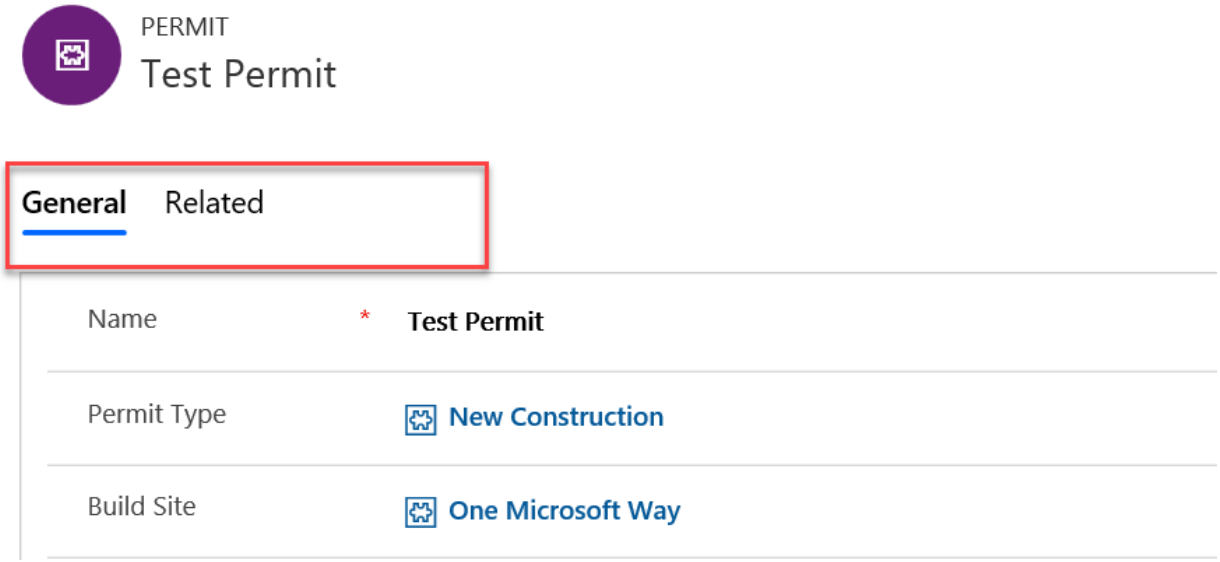
   - Select **Apps**.
   - Click to open the **Permit Management** application.
2. Open Permit record.
   - Select Permits from the Site Map.
   - Click to open a **Permit** record.



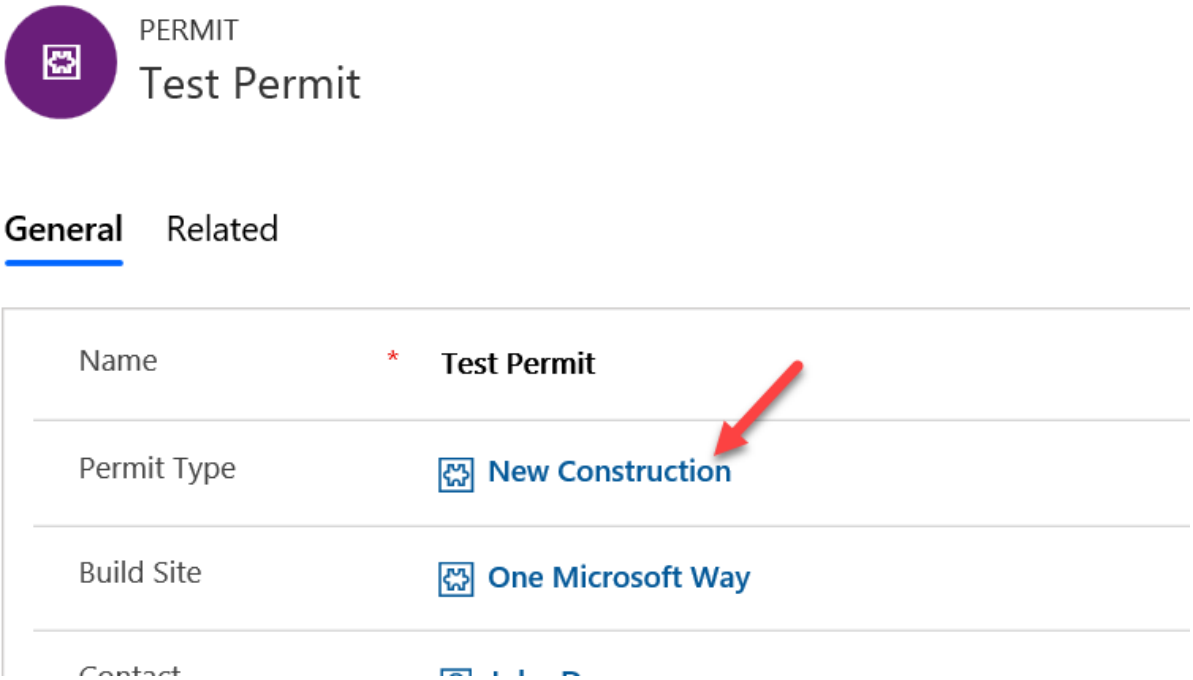
3. Check if the **Permit Type** Column is empty and if it is, the **Inspections** tab is hidden. In this case, the Permit Type is null.



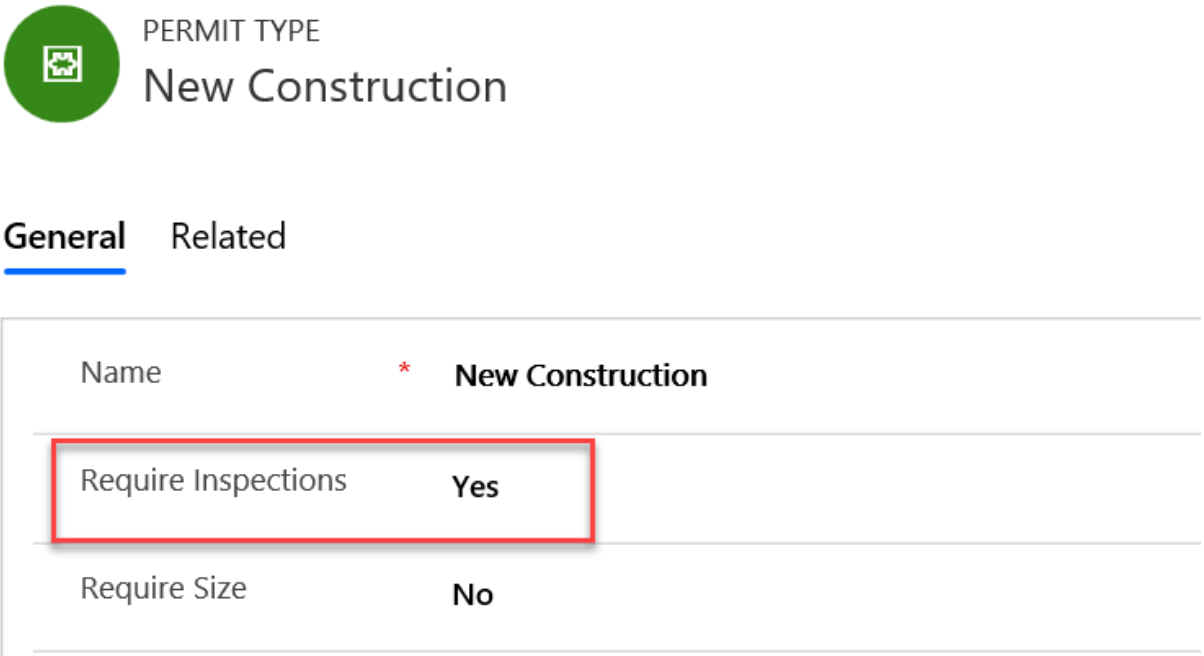
4. Select Permit Type.
   - Click on the **Permit Type** lookup.
   - Select **New Construction**.
   - Check if the **Inspections** tab is still hidden. If so, in this case, the Require Inspections Column value is false/No



5. Set **Require Inspections** Column value of the **Permit Type** to **Yes**.
   - Click on the selected **Permit Type**.

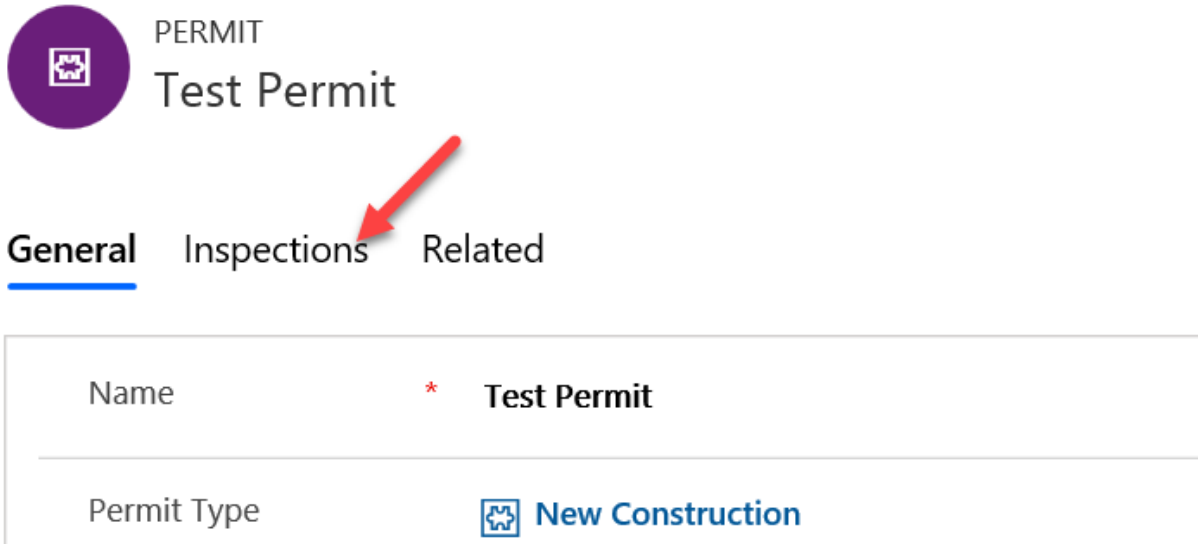


- Set the **Require Inspections** to **Yes**.

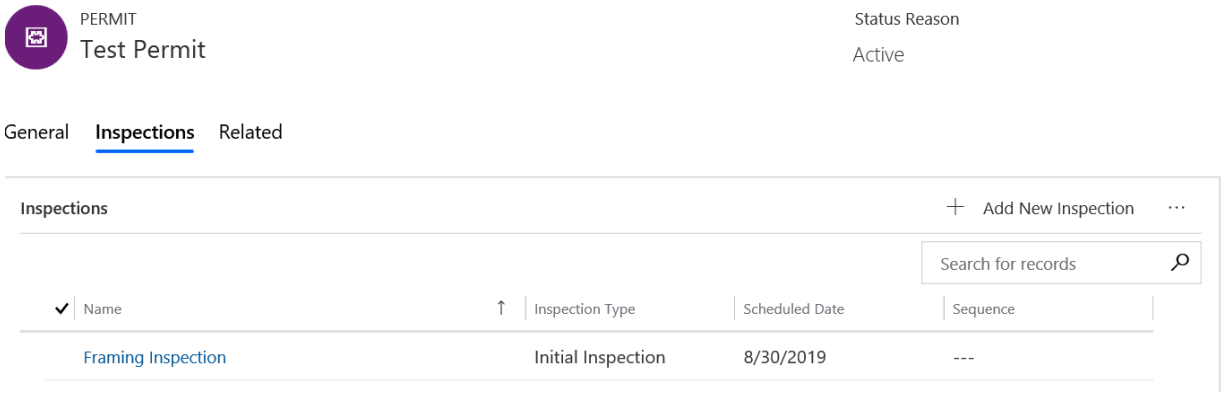


- Click **Save** button on the bottom right of the screen.
- Click on the browser back button.

6. You should now be able to see the Inspections tab.
   - Select the **Inspections** tab.



- The user should now be able to view/add inspections to the sub-grid.



20 Exercise #3: Toggle *required property on the Columns

Objective: In this exercise, you will create a script that will make the "New Size" Column required when the "Require Size" Column value is set to Yes. If the "Require Size" Column value is set to No, remove the requirement and make it optional. You will also hide the "New Size" Column. This logic will be driven by a Column on the permit type record that was retrieved using web API in the previous exercise.

20.1 Task #1: Create Function

1. Locate the _handlePermitTypeSettings function
   - Go back to **Visual Studio Code**.
   - Locate the **_handlePermitTypeSettings** function.
2. If permitType is null, remove the requirement and hide the "New Size" Column.
   - Add the script mentioned below in the **if permitType == null** statement. contoso_newsize is the logical name of the New Size Column.

```
formContext.getAttribute("contoso_newsize").setRequiredLevel("none");

formContext.ui.controls.get("contoso_newsize").setVisible(false);
```

```
},
_handlePermitTypeSettings: function (executionContext) {
    var formContext = executionContext.getFormContext();
    var permitType = formContext.getAttribute("contoso_permittype").getValue();
    if (permitType == null) {
        formContext.ui.tabs.get("inspectionsTab").setVisible(false);
        formContext.getAttribute("contoso_newsize").setRequiredLevel("none");
        formContext.ui.controls.get("contoso_newsize").setVisible(false);
        return;

    } else {
```

3. Check if “Require Size” Column value of the Permit Type is set to Yes
   - Add the script mentioned below inside the retrieveRecord function.

```
if (result.contoso_requiresize) {

} else {

}
```

```
        return;

    } else {
        var permitTypeID = permitType[0].id;
        Xrm.WebApi.retrieveRecord("contoso_permittype", permitTypeID).then(function (result) {
            if (result.contoso_requireinspections) {
                formContext.ui.tabs.get("inspectionsTab").setVisible(true);
            } else {
                formContext.ui.tabs.get("inspectionsTab").setVisible(false);
            }

            if (result.contoso_requiresize) {

            } else {

            }


        },
        function (error) { alert('Error:' + error.message) });
```

4. If “Require Size” Column value of the Permit Type is set to Yes, make the “New Size” Column visible and as required.
   - Add the script mentioned below in the **if result.contoso_requiresize** statement. contoso_requiresize is the logical name of the Require Size Column.

```
formContext.ui.controls.get("contoso_newsize").setVisible(true);

formContext.getAttribute("contoso_newsize").setRequiredLevel("required");
```

```
} else {
    var permitTypeID = permitType[0].id;
    Xrm.WebApi.retrieveRecord("contoso_permittype", permitTypeID).then(function (result) {
        if (result.contoso_requireinspections) {
            formContext.ui.tabs.get("inspectionsTab").setVisible(true);
        } else {
            formContext.ui.tabs.get("inspectionsTab").setVisible(false);
        }

        if (result.contoso_requiresize) {
            formContext.ui.controls.get("contoso_newsize").setVisible(true);
            formContext.getAttribute("contoso_newsize").setRequiredLevel("required");

        } else {

        }


    },
```

5. If Require Size Column value of the Permit Type is not set to Yes, remove the “New Size” Column not required and hide it.
   - Add the script mentioned below inside the else statement.

     ```
     formContext.getAttribute("contoso_newsize").setRequiredLevel("none");

     formContext.ui.controls.get("contoso_newsize").setVisible(false);
     ```

```
    Xrm.WebApi.retrieveRecord("contoso_permittype", permitTypeID).then(function (result) {
        if (result.contoso_requireinspections) {
            formContext.ui.tabs.get("inspectionsTab").setVisible(true);
        } else {
            formContext.ui.tabs.get("inspectionsTab").setVisible(false);
        }

        if (result.contoso_requiresize) {
            formContext.ui.controls.get("contoso_newsize").setVisible(true);
            formContext.getAttribute("contoso_newsize").setRequiredLevel("required");

        } else {
            formContext.getAttribute("contoso_newsize").setRequiredLevel("none");
            formContext.ui.controls.get("contoso_newsize").setVisible(false);
        }
```

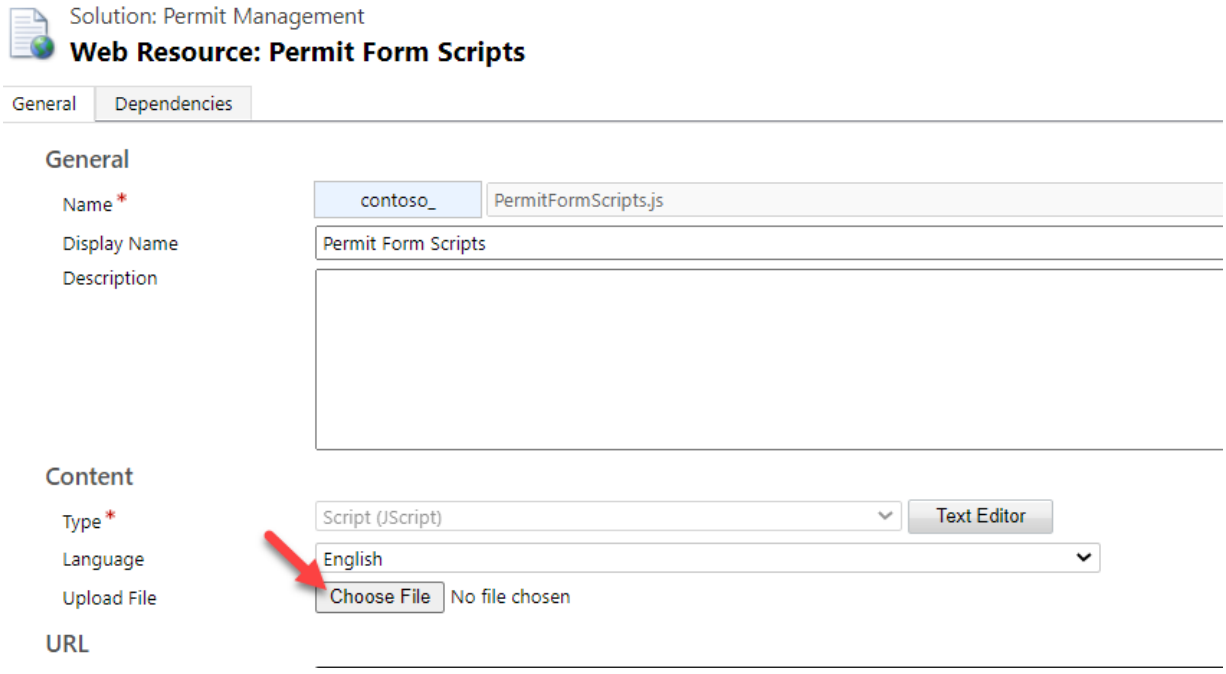
6. The _handlePermitTypeSettings function should now look like the image below.

```
_handlePermitTypeSettings: function (executionContext) {
    var formContext = executionContext.getFormContext();
    var permitType = formContext.getAttribute("contoso_permittype").getValue();
    if (permitType == null) {
        formContext.ui.tabs.get("inspectionsTab").setVisible(false);
        formContext.getAttribute("contoso_newsize").setRequiredLevel("none");
        formContext.ui.controls.get("contoso_newsize").setVisible(false);
        return;

    } else {
        var permitTypeID = permitType[0].id;
        Xrm.WebApi.retrieveRecord("contoso_permittype", permitTypeID).then(function (result)
            if (result.contoso_requireinspections) {
                formContext.ui.tabs.get("inspectionsTab").setVisible(true);
            } else {
                formContext.ui.tabs.get("inspectionsTab").setVisible(false);
            }

            if (result.contoso_requiresize) {
                formContext.ui.controls.get("contoso_newsize").setVisible(true);
                formContext.getAttribute("contoso_newsize").setRequiredLevel("required");
            } else {
                formContext.getAttribute("contoso_newsize").setRequiredLevel("none");
                formContext.ui.controls.get("contoso_newsize").setVisible(false);
            }
        },
        function (error) { alert('Error:' + error.message) });
    }
```

Click **File** and **Save All**.

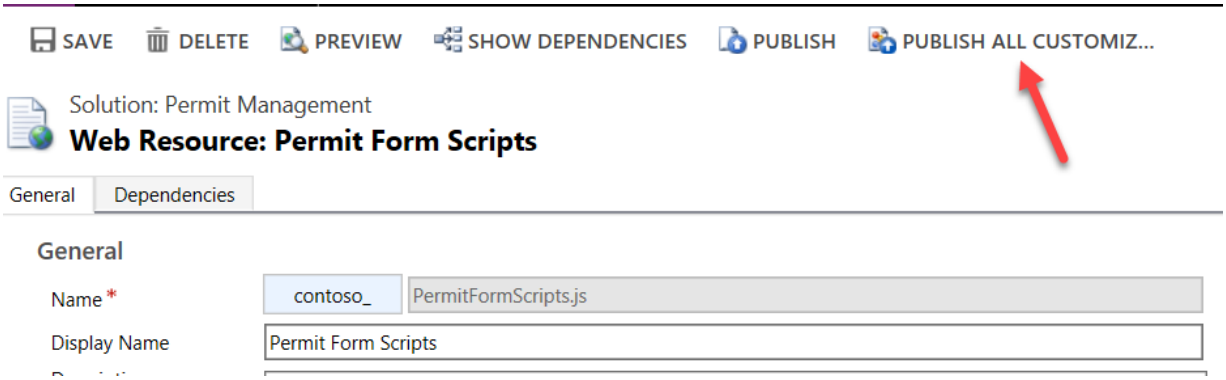
20.2 Task #2: Load Updated Script

1. Open the Permit Form Script web resource.
    - Go back to the Permit Form Scripts web resource.
    - Click **Choose File**.



    - Select the **PermitFormFunctions.js** you updated and click **Open**.
2. Save and Publish your changes

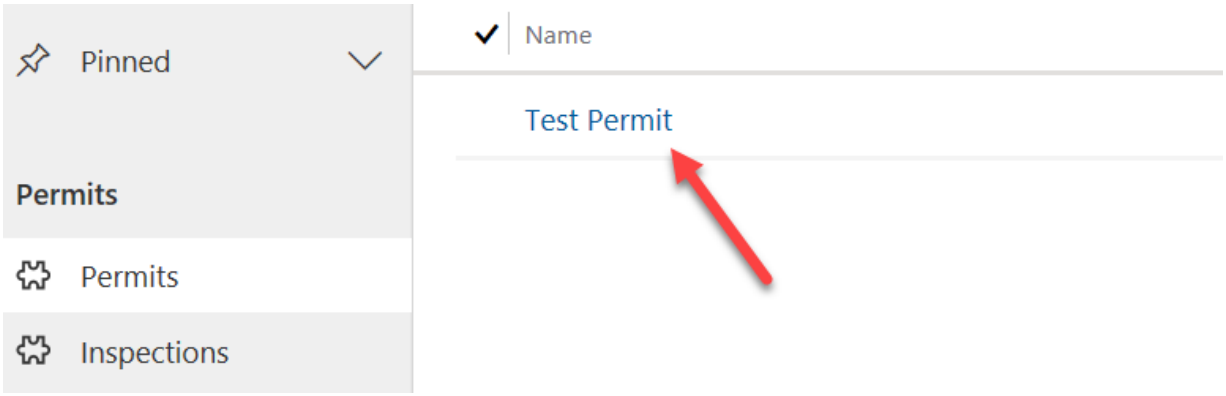
- Click **Save** and wait until the changes are saved.
- Click **Publish All Customizations** and wait for the publishing to complete.



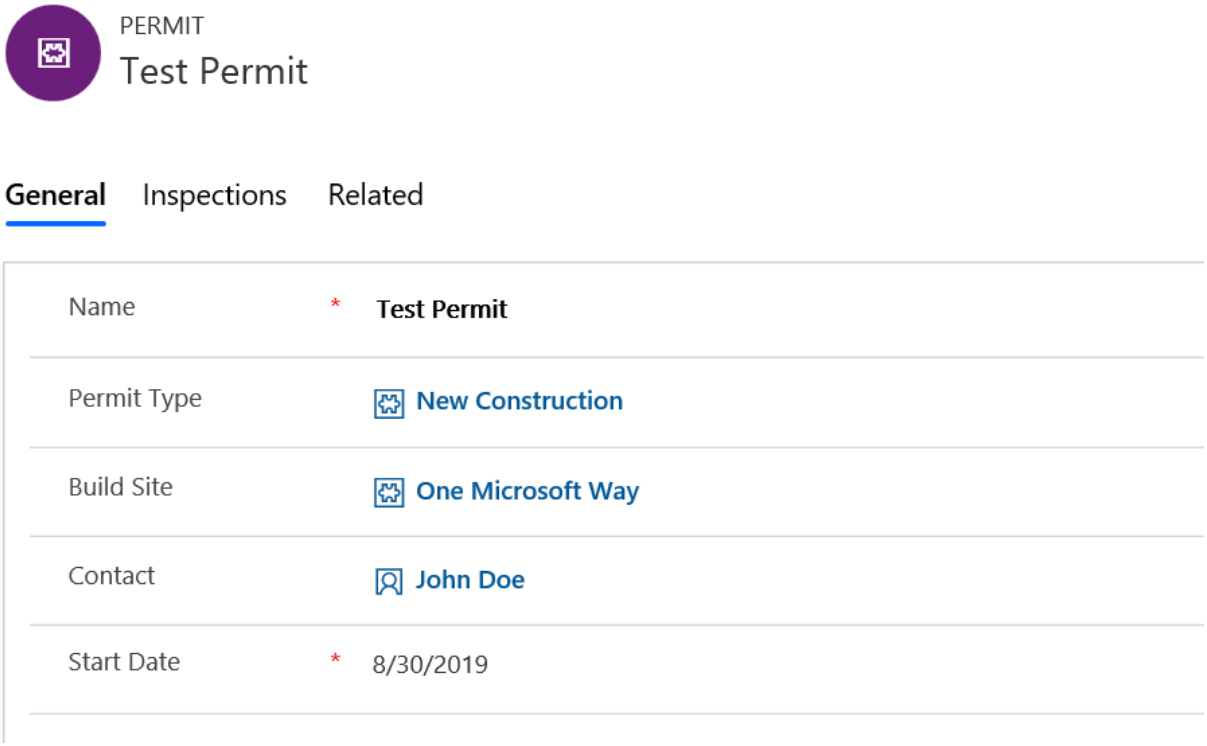
Close the Web Resource editor.

20.3 Task #3: Test Your Changes

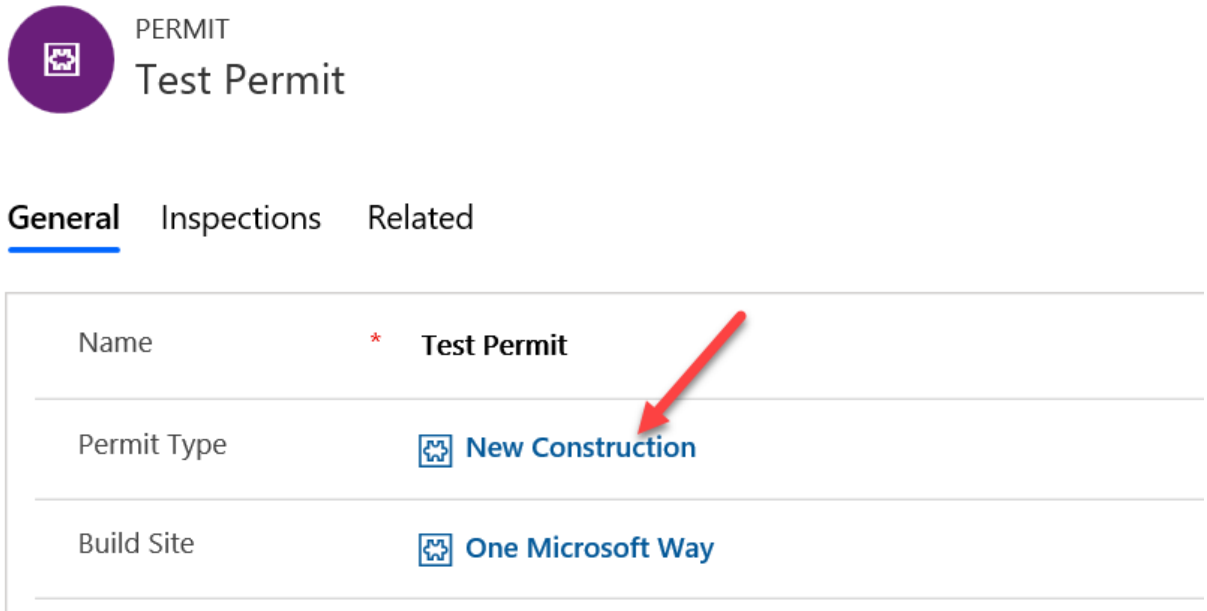
1. Start the Permit Management application
   - Log on to https://make.powerapps.com and select your **Dev** environment.
   - Select **Apps**.
   - Click to open the **Permit Management** application.
2. Open Permit record.
   - Select Permits.
   - Click to open a **Permit** record.



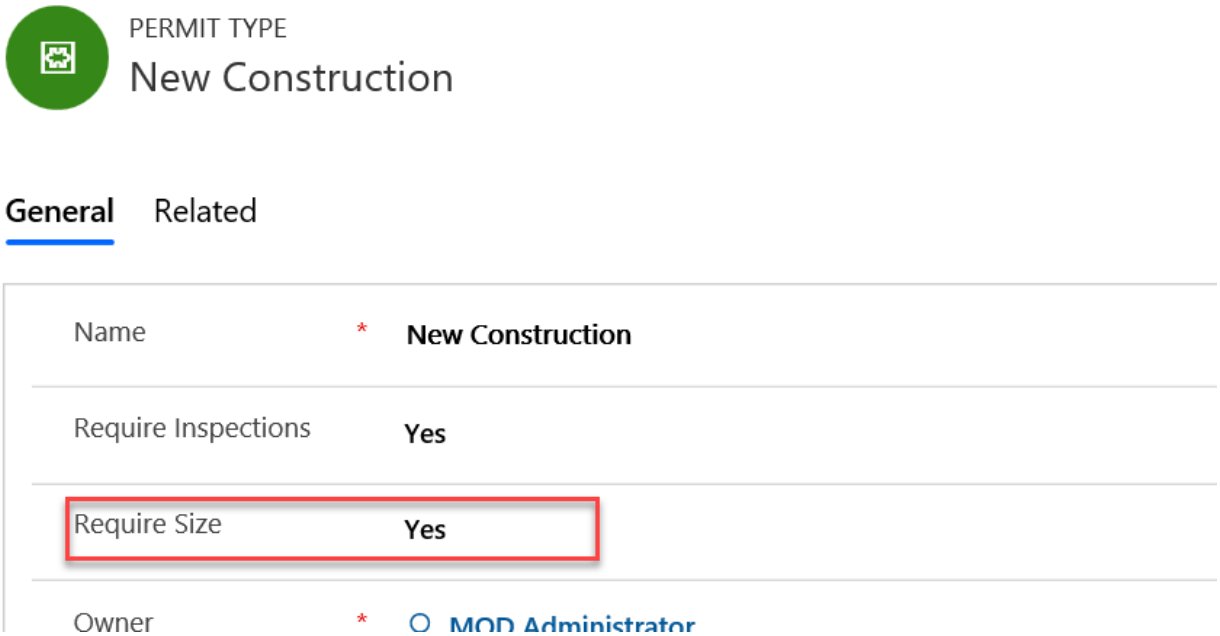
3. Check if the **New Size** Column is hidden. If so, then it is because in this case, the "Require Size" Column of the Permit Type is set to NO.



4. Set **Require Size** Column value of the **Permit Type** to **Yes**.
   - Click on the selected **Permit Type**.

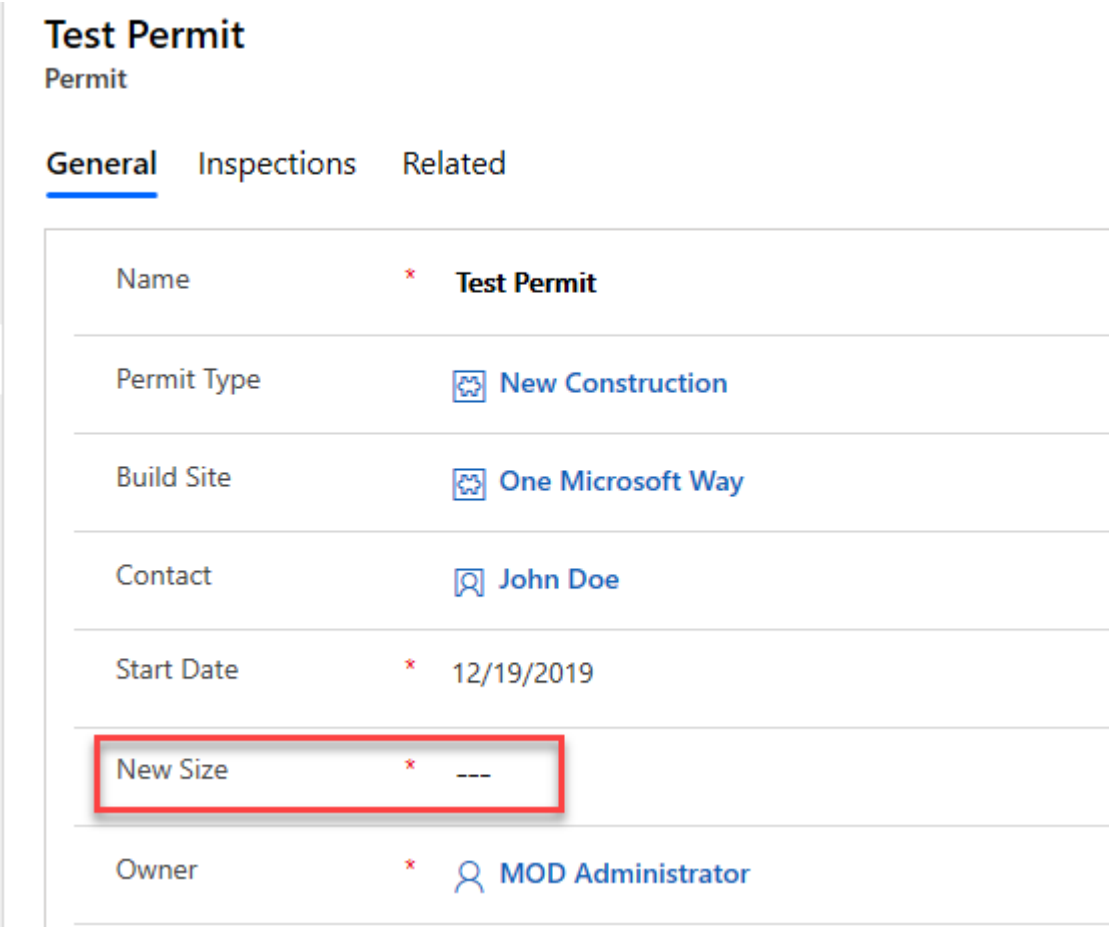


   - Set the **Require Size** to **Yes**.

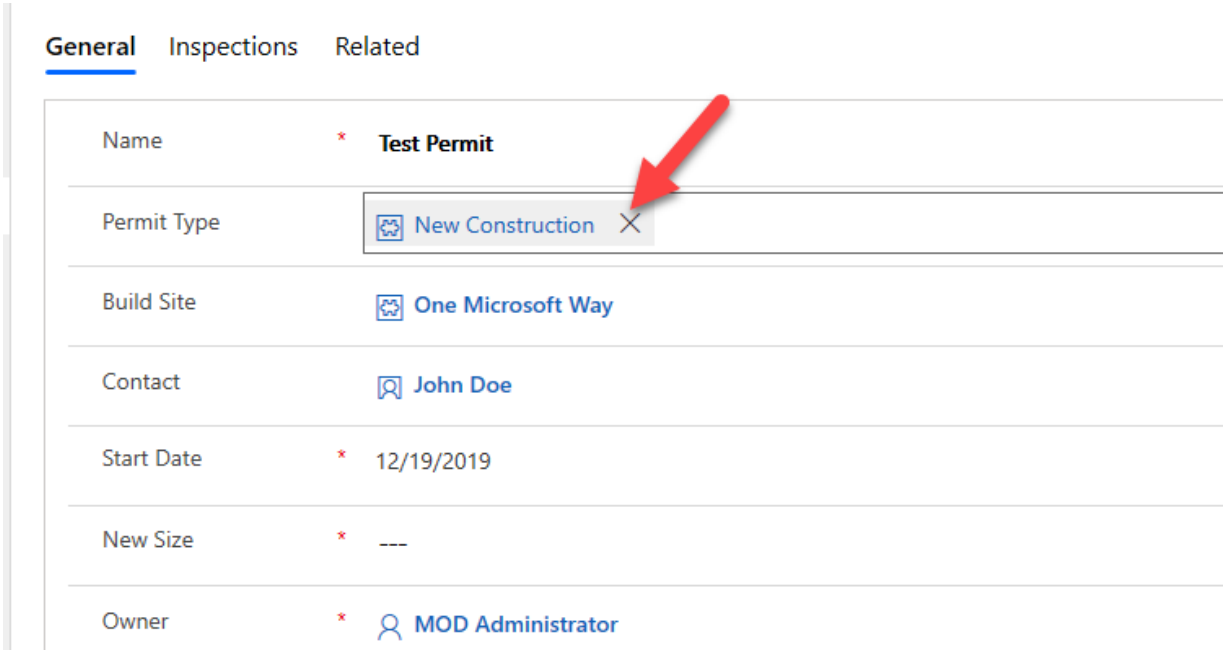


- Click **Save** on the bottom right of the screen.
- Click on the browser back button.

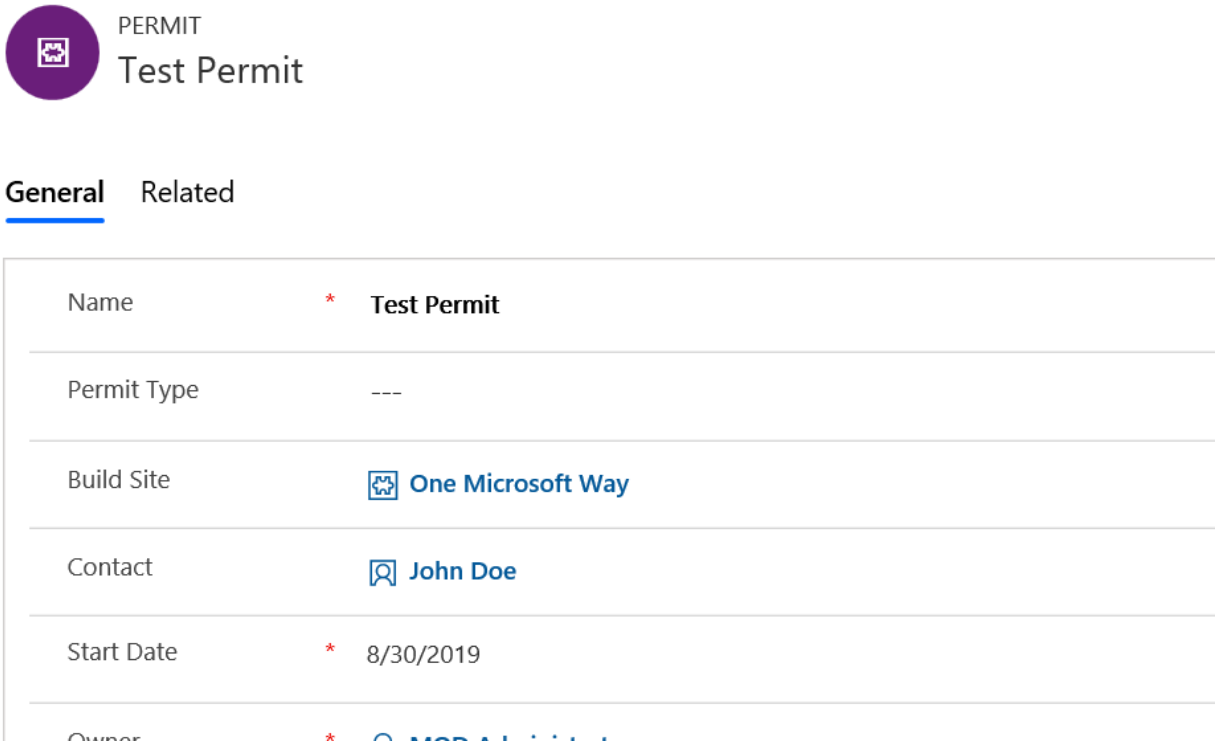
5. Check if the "New Size" Column is visible and it is marked as required.
   - You should now be able to see "**New Size"** Column on the form and this is marked as required.



- Remove **Permit Type**.



- Check if both the **Inspections** tab and **New Size** Column are now hidden. They should be removed as soon as the "Permit Type" is removed.



21 Exercise #4: Command Button Function

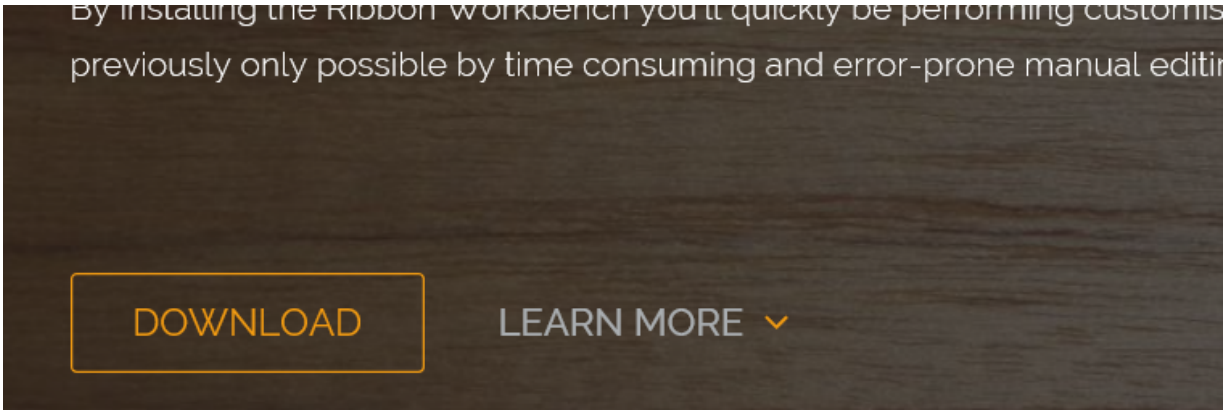
Objective: In this exercise, you will download and install the Ribbon Workbench tool to edit the command bar. Through this, you will also create action, create function that will lock permits, add a button to the permit Table and call the lock permit function when the button is clicked.

21.1 Task #1: Download and Install Ribbon Workbench

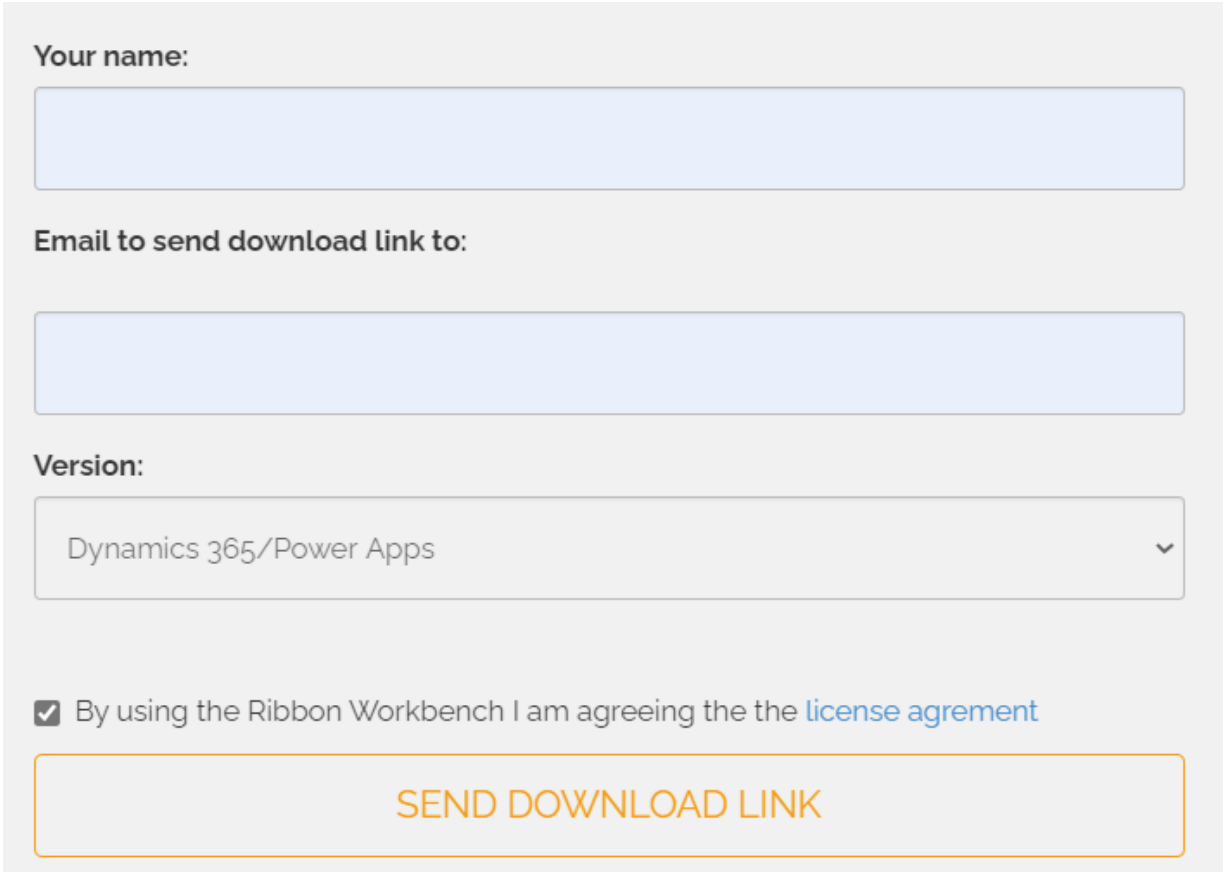
Ribbon Workbench is a community tool that makes it easier to edit the command bar on a form. Alternatively, you can do this without the tool, by directly editing the RibbonDiffXml.

1. Get Ribbon Workbench download link

- Log on to [http://www.develop1.net/public/Download%20Ribbon%20Workbench%202013.aspx](http://www.deve
Ribbon Workbench 2013.aspx)
- Click **Download**.

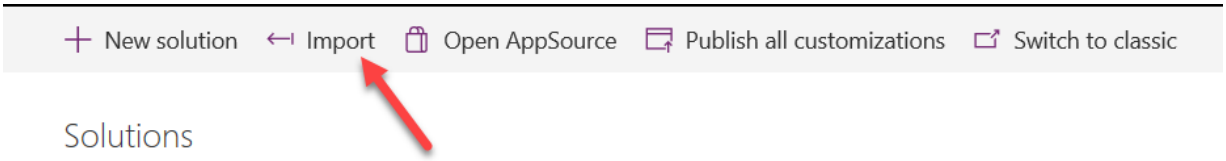


- Provide a name and an email and the download link will be sent to that email.
- Select **Dynamics 365/Power Apps** for **Version** available and click **Send Download Link**.

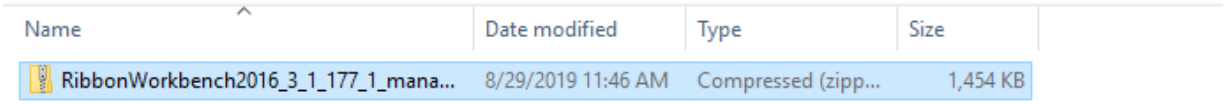


- Wait to receive the download link in an email to the provided email address.

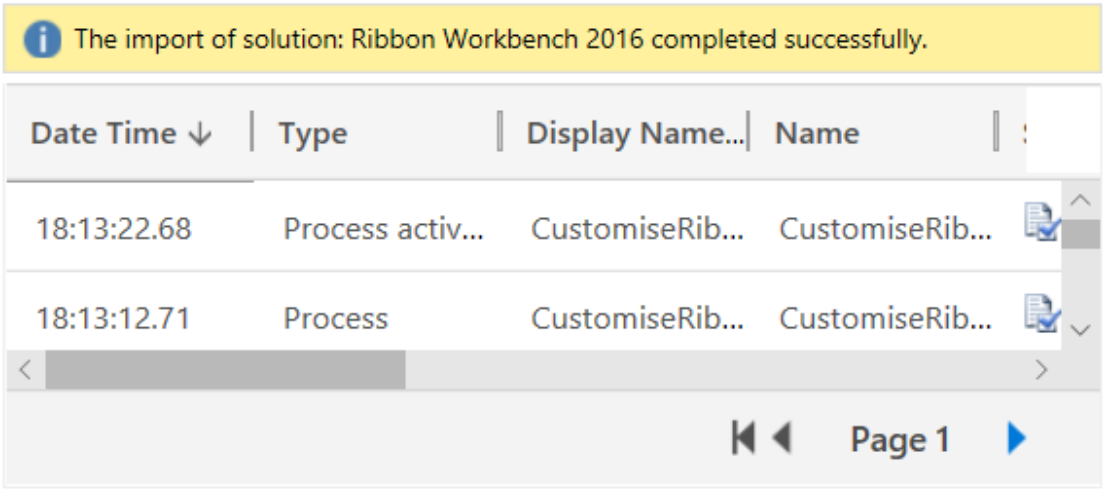
2. Download Ribbon Workbench
   - After you receive the email, click on the download button.
   - Save the downloaded solution on your machine.
3. Install Ribbon Workbench solution.
   - Sign in to Power Apps maker portal and select your **Dev** environment.
   - Select **Solutions** and click **Import**.

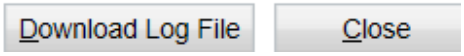


- Click **Choose File**.
- Select the **RibbonWorkbench** solution you just downloaded and click **Open**.

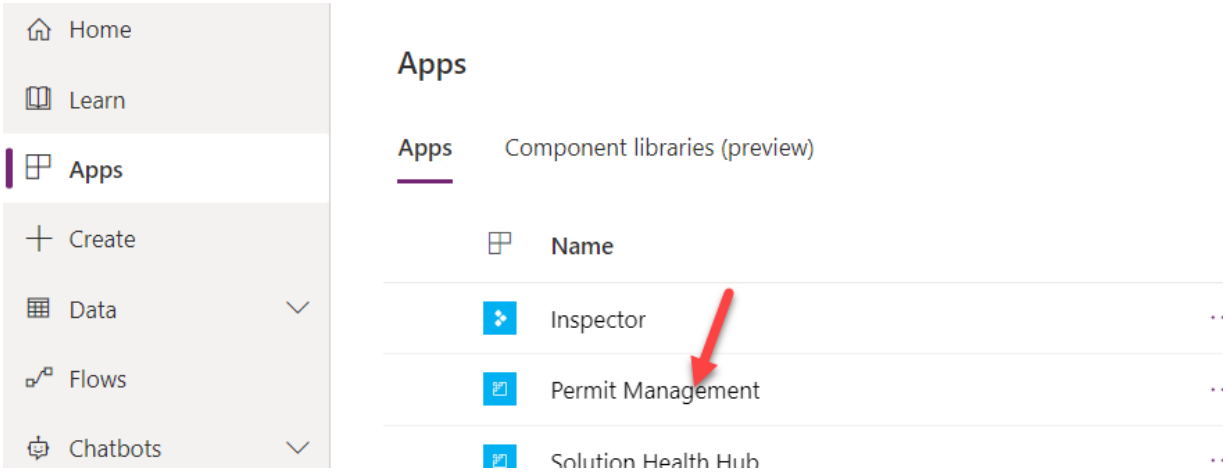


- Click **Next**.
- Click **Next** again.
- Click **Import** and wait for the import to complete.
- Click **Close**.

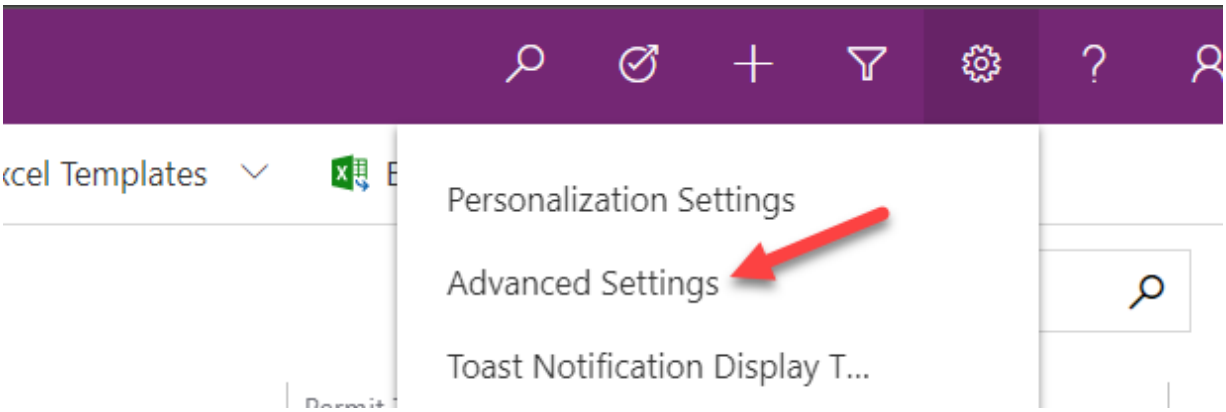




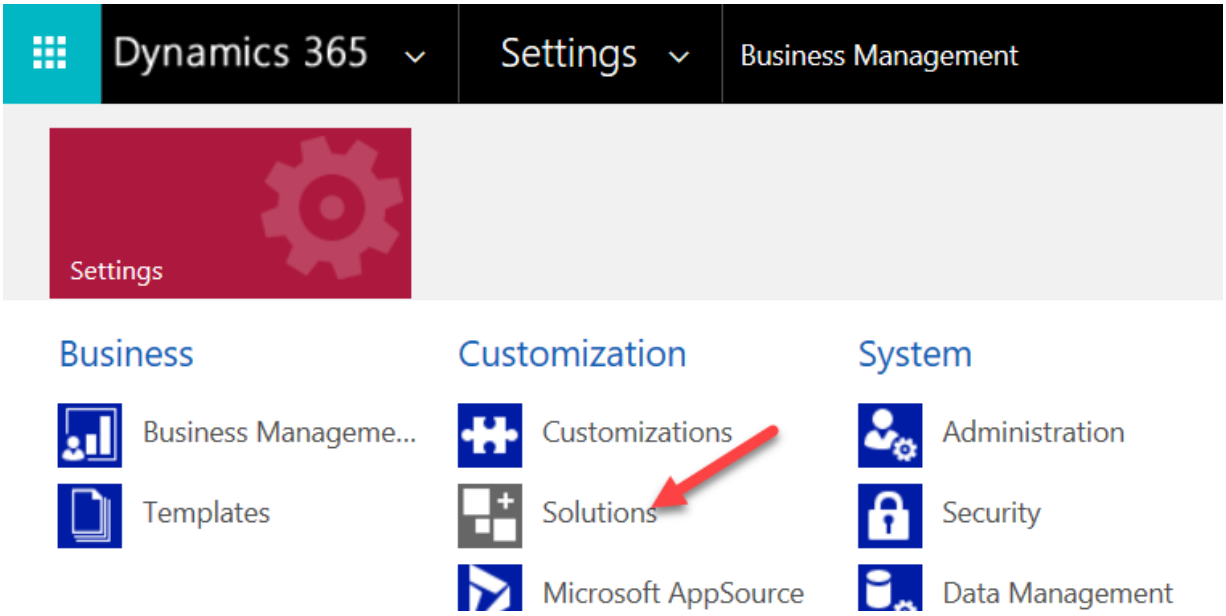
4. Confirm the Ribbon Workbench was installed
   - Select **Apps** and click to open the **Permit Management** application.



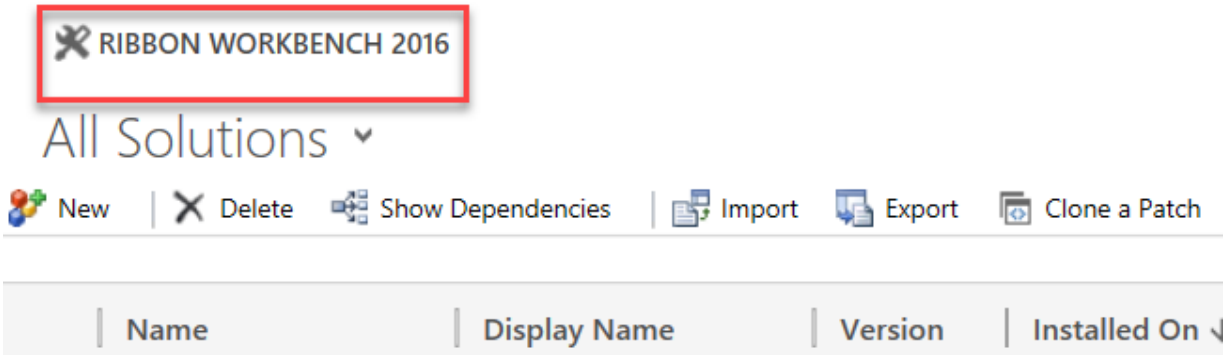
- Click **Settings** and select **Advanced Settings**.



- Click **Settings** | **Solutions**.



- You should see the Ribbon Workbench button on the top.

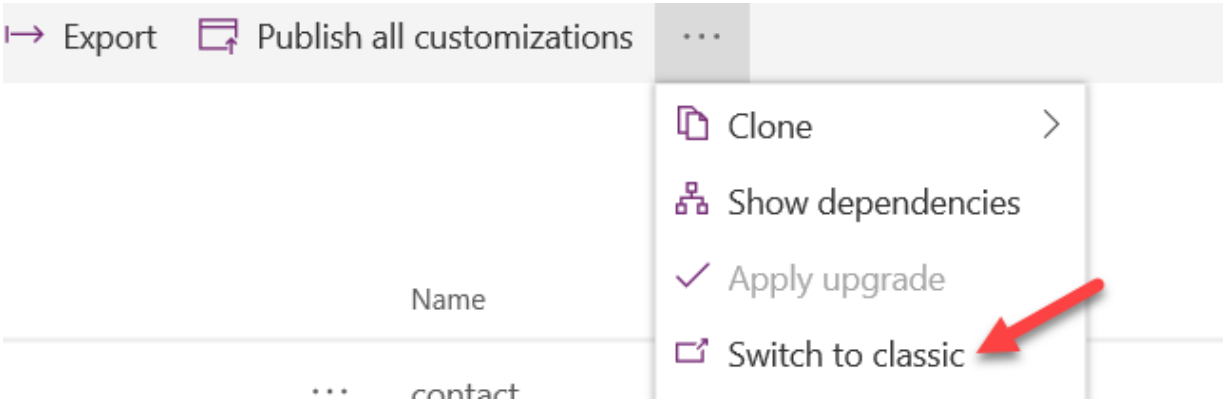


5. Close the **Advanced Settings** browser tab or window.
6. Close the Permit Management application.

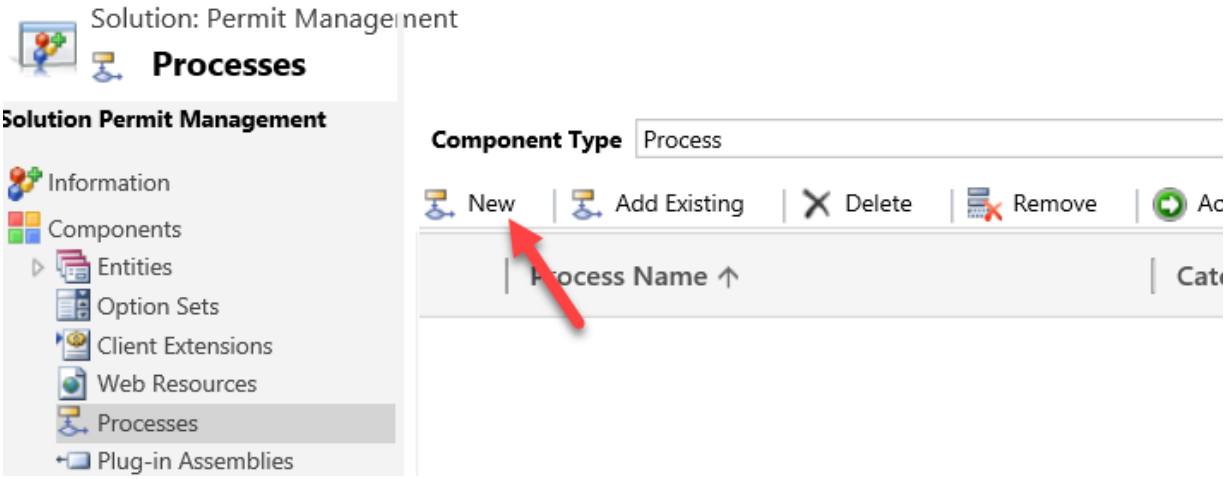
21.2 Task #2: Create Action Process

In this task, you will create a custom action that will be called to lock the permit. You will not be implementing the business logic in this lab to lock the permit. It will be completed later in the class when you build the plug-in that registers on the custom actions you will define here.

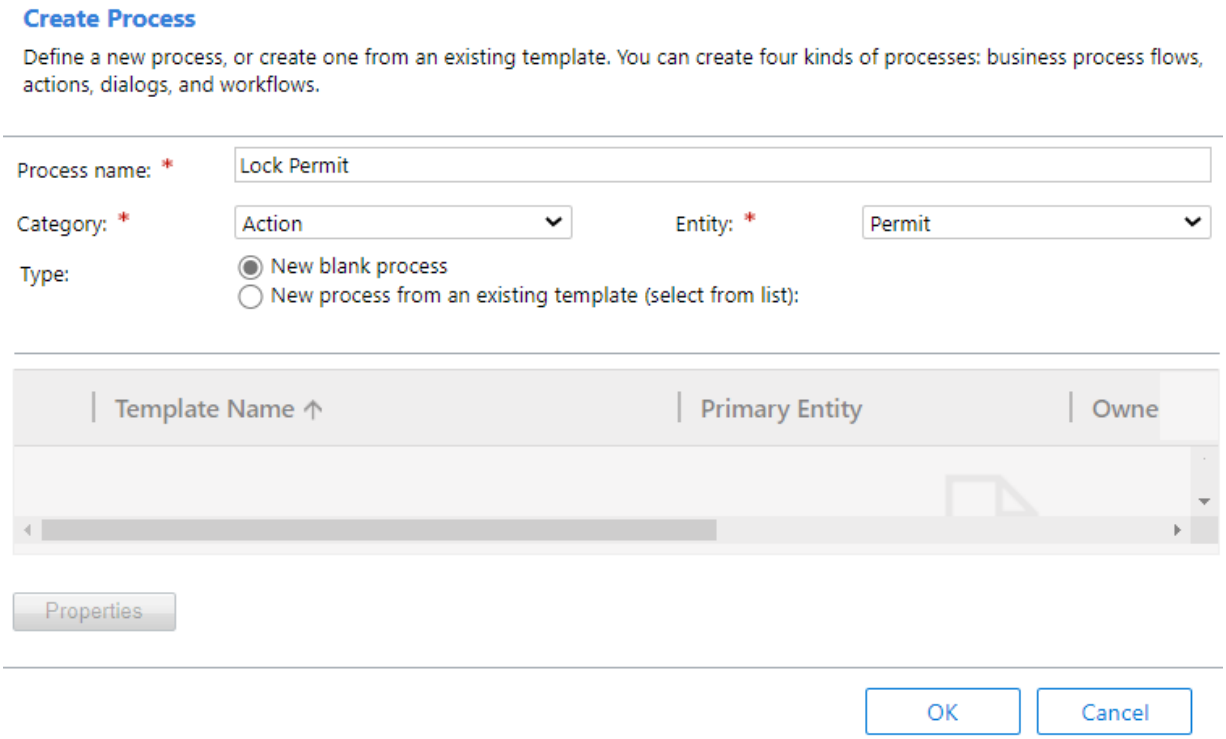
1. Open the Permit Management Solution.
   - Navigate to Power Apps makes portal and make sure you have the **Dev** environment selected.
   - Select **Solutions** and click to open the **Permit Management** solution.
2. Switch to Classic
   - Click on the **....** button and select **Switch to Classic**.



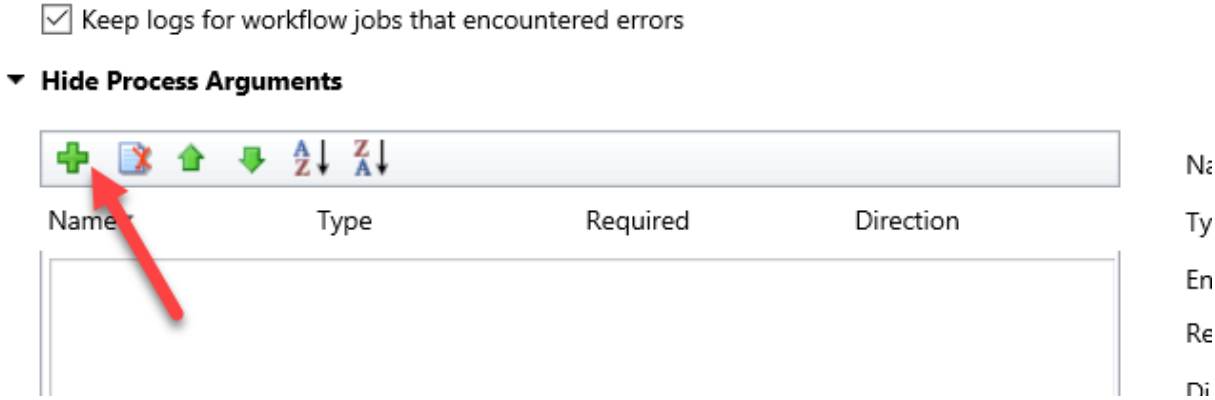
3. Create new process
   - Select **Processes** and click **New**.



- Enter **Lock Permit** for **Name**, select **Action** from the dropdown for **Category**, select **Permit** from the dropdown for **Table**, select **New Blank Process** for **Type**, and click **OK**.



4. Add process arguments
   - Go to the **Process Arguments** section and click **Add**.

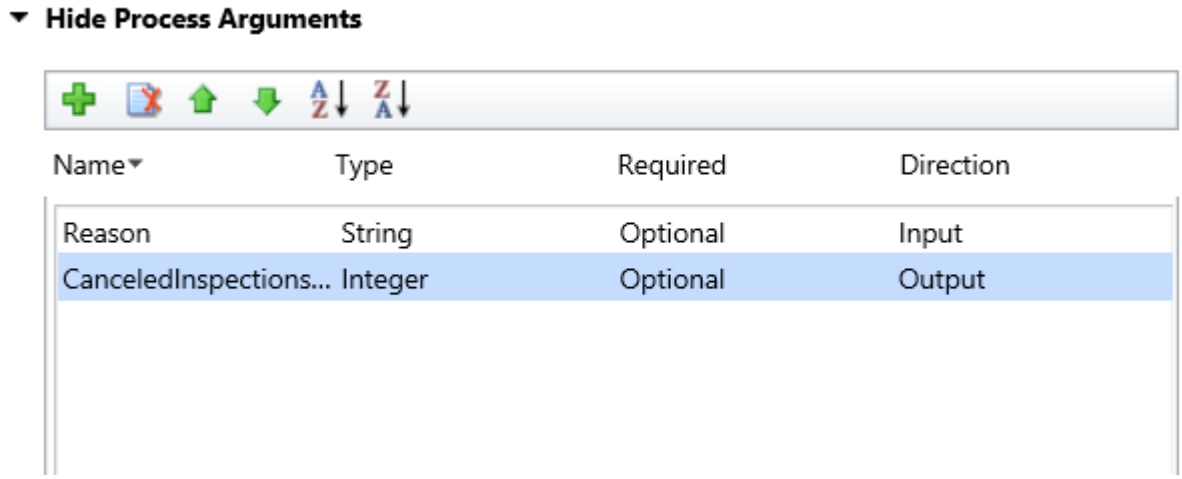


   - Enter **Reason** for **Name**, select **Type as String**, and select **Input** for **Direction**.

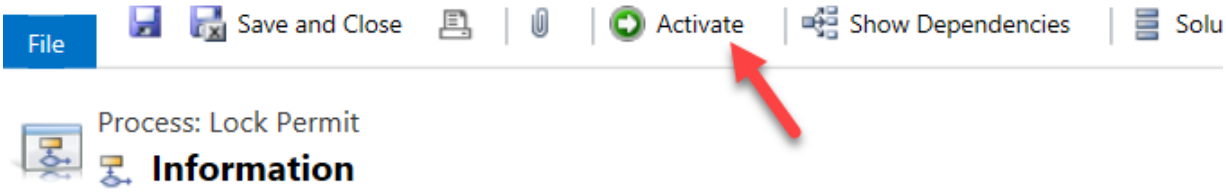
   - Click **Add** again.
   - Enter **CanceledInspectionsCount** for **Name**, select **Type** as **Integer**, and select **Output** for

Direction.

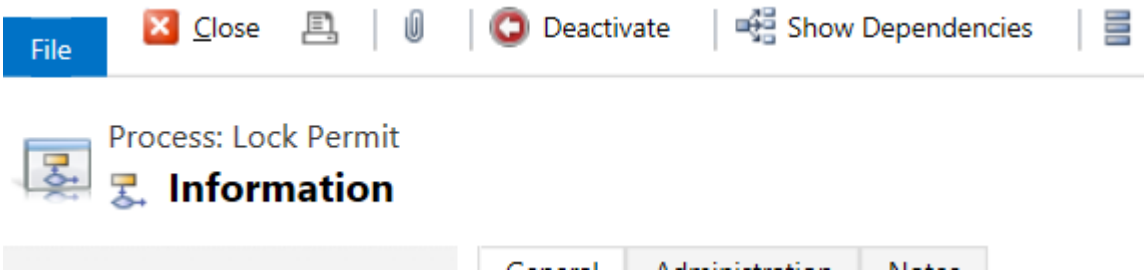
- You should now have one input and one output arguments.



5. Save and activate action
   - Click **Save** and wait until the changes are saved.
   - Click **Activate**. This will open a pop-up for confirmation. Click **Activate.**



   - Confirm the activation and wait for the action to be activated. You should now be able to see the Deactivate option in the Top menu.
   - Click **Close**.

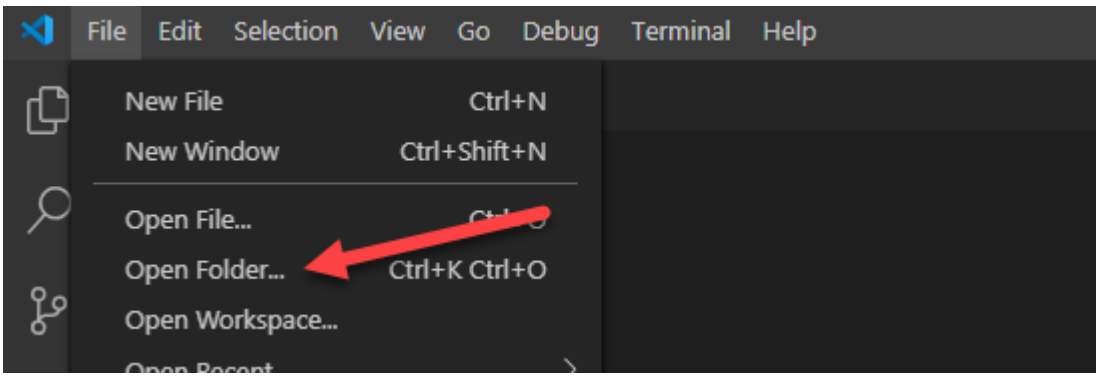


   - Close the solution explorer.

21.3 Task #3: Create the Function

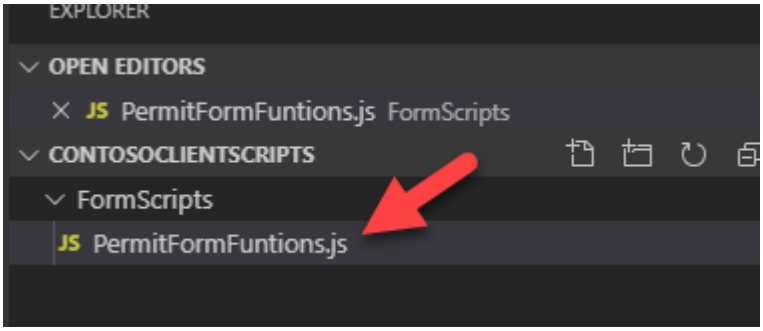
In this task, you will create the logic to invoke a custom action using the web API.

1. Start Visual Studio Code and open the resources you create in exercise one
   - Start **Visual Studio Code**.
   - Click **File** and select **Open Folder**.



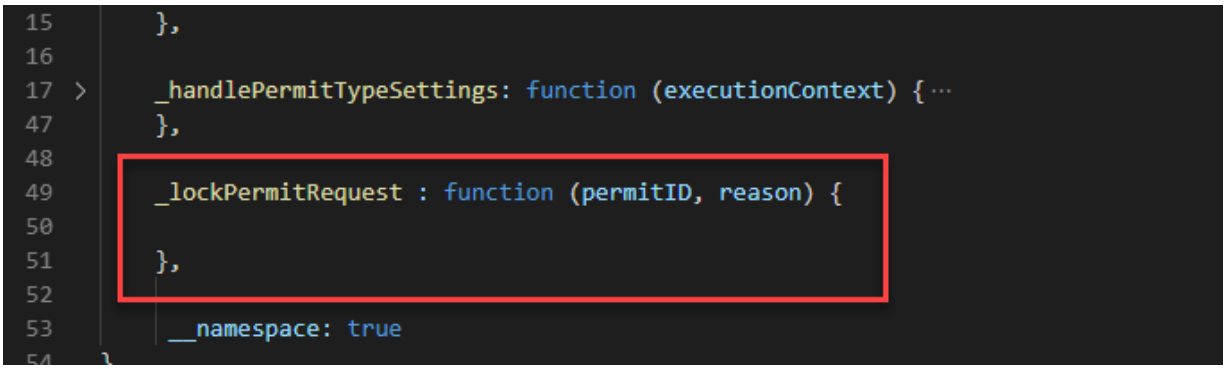
- Select the **ContosoClientScrpts** folder you created in exercise one and click **Select Folder**.

2. Add a function that will build the request
   - Open the **PermitFormFunctions.js** file.



   - Add the function below after the _handlePermitTypeSettings function.

```
_lockPermitRequest : function (permitID, reason) {

},
```



3. Build entity and set reason.
   - Add the script mentioned below inside the **__lockPermitRequest** function.

```
this.entity = { entityType: "contoso_permit", id: permitID };

this.Reason = reason;
```

4. Build and return the request

- Add the script mentioned below in the **__lockPermiRequest** function.

```
this.getMetadata = function () {
    return {
        boundParameter: "entity", parameterTypes: {
            "entity": {
                typeName: "mscrm.contoso_permit",
                structuralProperty: 5
            },
            "Reason": {
                "typeName": "Edm.String",
                "structuralProperty": 1 // Primitive Type
            }
        },
        operationType: 0, // This is an action. Use '1' for functions and '2' for CRUD
        operationName: "contoso_LockPermit",
    };
};
```

```
_lockPermitRequest: function (permitID, reason) {
    this.entity = { entityType: "contoso_permit", id: permitID };
    this.Reason = reason;
    this.getMetadata = function () {
        return {
            boundParameter: "entity", parameterTypes: {
                "entity": {
                    typeName: "mscrm.contoso_permit",
                    structuralProperty: 5
                },
                "Reason": {
                    "typeName": "Edm.String",
                    "structuralProperty": 1 // Primitive Type
                }
            },
            operationType: 0, // This is an action. Use '1' for functions and '2' for CRUD
            operationName: "contoso_LockPermit",
        };
    };
},
```

5. Add the function that will be called from the action button.
   - Add the function mentioned below after the **__lockPermitRequest** function.

```
lockPermit: function (primaryControl) {

},
```

```
66      },
67
68      lockPermit: function (primaryControl) {
69
70      },
71
72      __namespace: true
73  }
```

6. Get Permit ID and call **__lockPermitRequest**
   - Get the id by adding the script mentioned below inside the **lockPermit** function.

```
formContext = primaryControl;
var PermitID = formContext.data.entity.getId().replace('{', '').replace('}', '');
```

- Call **__lockPermitRequest**. We are hardcoding the reason “Admin Lock”

```
var lockPermitRequest = new ContosoPermit.Scripts.PermitForm._lockPermitRequest(PermitID, "A
```

7. Execute the request.
   - Add the script mentioned below inside the lockPermit function.

```
// Use the request object to execute the function
    Xrm.WebApi.online.execute(lockPermitRequest).then(
        function (result) {
            if (result.ok) {
                console.log("Status: %s %s", result.status, result.statusText);
                // perform other operations as required;
                formContext.ui.setFormNotification("Status " + result.status, "INFORMATION")
            }
        },
        function (error) {
            console.log(error.message);
            // handle error conditions
        }
    );
```

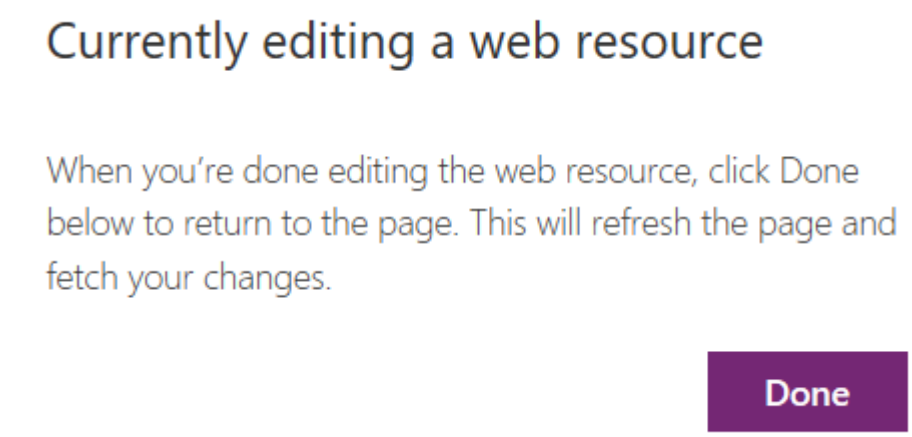
```
},

lockPermit: function (primaryControl) {
    formContext = primaryControl;
    var PermitID = formContext.data.entity.getId().replace('{', '').replace('}', '');

    var lockPermitRequest = new ContosoPermit.Scripts.PermitForm._lockPermitRequest(PermitID, "Admin Lock");

    // Use the request object to execute the function
    Xrm.WebApi.online.execute(lockPermitRequest).then(
    function (result) {
        if (result.ok) {
            console.log("Status: %s %s", result.status, result.statusText);
            // perform other operations as required;
            formContext.ui.setFormNotification("Status " + result.status, "INFORMATION");
        }
    },
    function (error) {
        console.log(error.message);
        // handle error conditions
    });
},

  namespace: true
```

   - Click **File** and **Save All**.

8. Load the update resource and publish.
   - Log on to Power Apps maker portal and make sure you have the **Dev** environment selected.
   - Select **Solutions** and click to open the **Permit Management** solution.
   - Click to open the **Permit Form Scripts** web resource. This will open a new browser window.

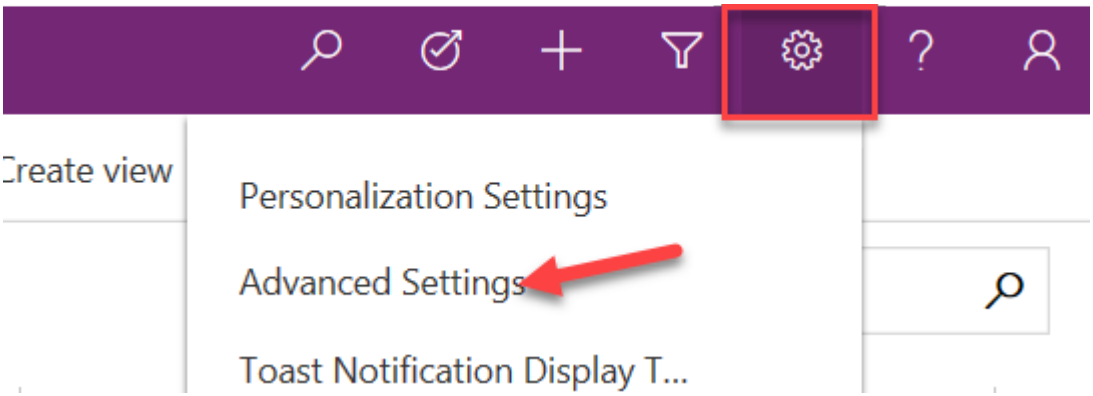
   - Click **Choose File**.
   - Select the **PermitFormFunctions.js** you updated and click **Open**.
   - Click **Save** and wait until the changes are saved.
   - Click **Publish All Customizations** and wait for the publishing to complete.

- Close the web resource editor and return to the solutions window.
- Click **Done**.

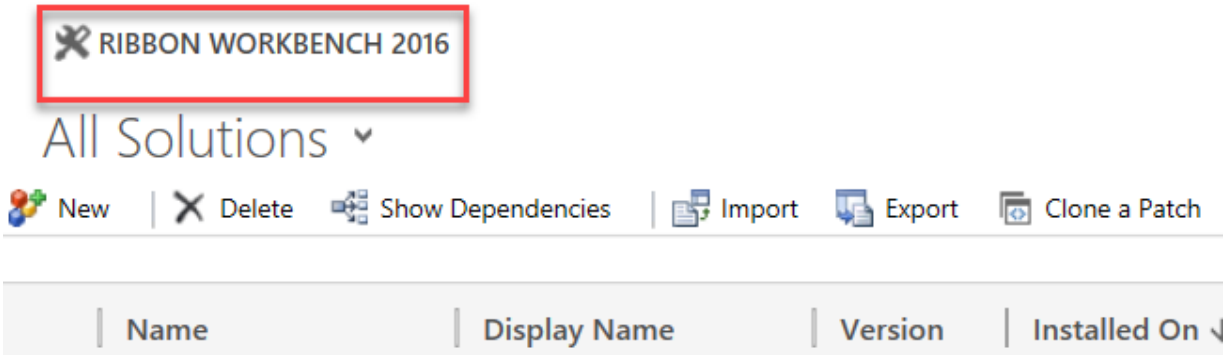


21.4 Task #4: Add Button to Ribbon

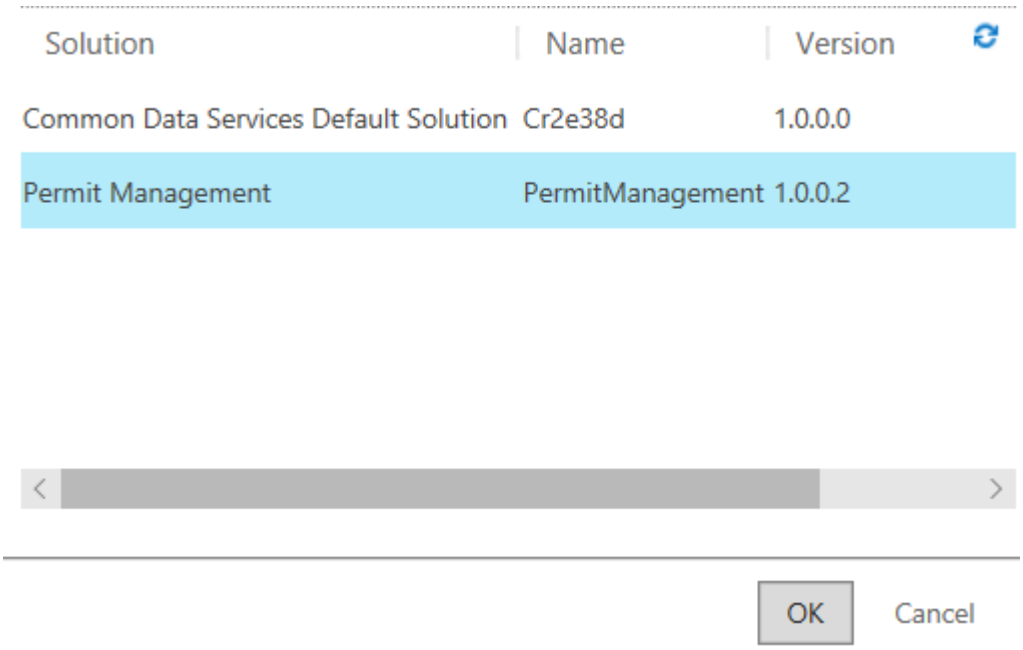
1. Open Ribbon Workbench
   - Sign in to Power Apps maker portal and make sure you have the **Dev** environment selected.
   - Select **Apps** and click to open the **Permit Management** application.
   - Click **Settings** and select **Advanced Settings**.



   - Click **Settings** | **Solutions**.
   - Click on the Ribbon Workbench button.

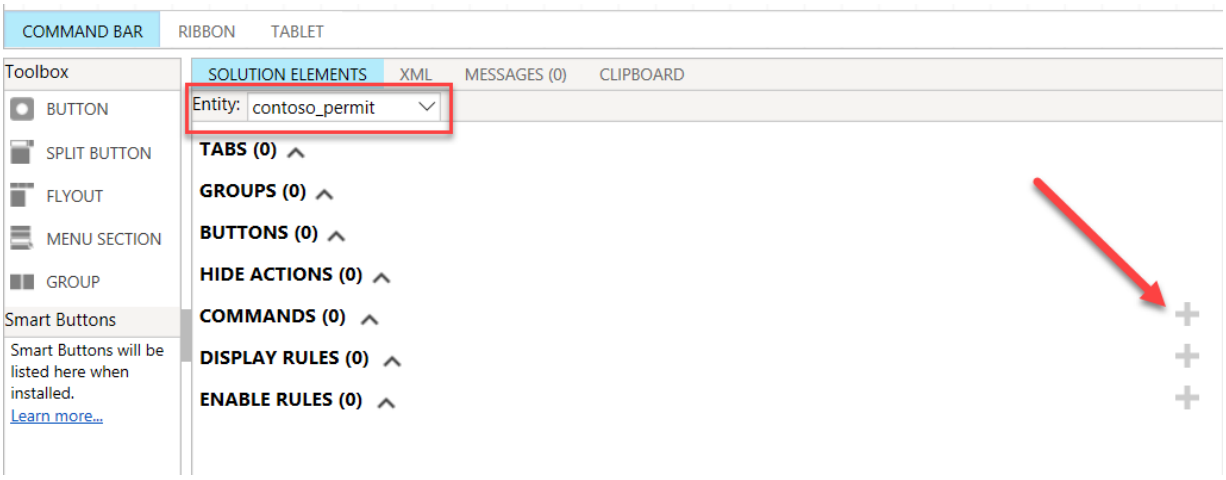


2. Load the Permit Management solution
   - Select the **Permit Management** solution in the pop-up opened for selecting the Load Solution.
   - Click **OK**.

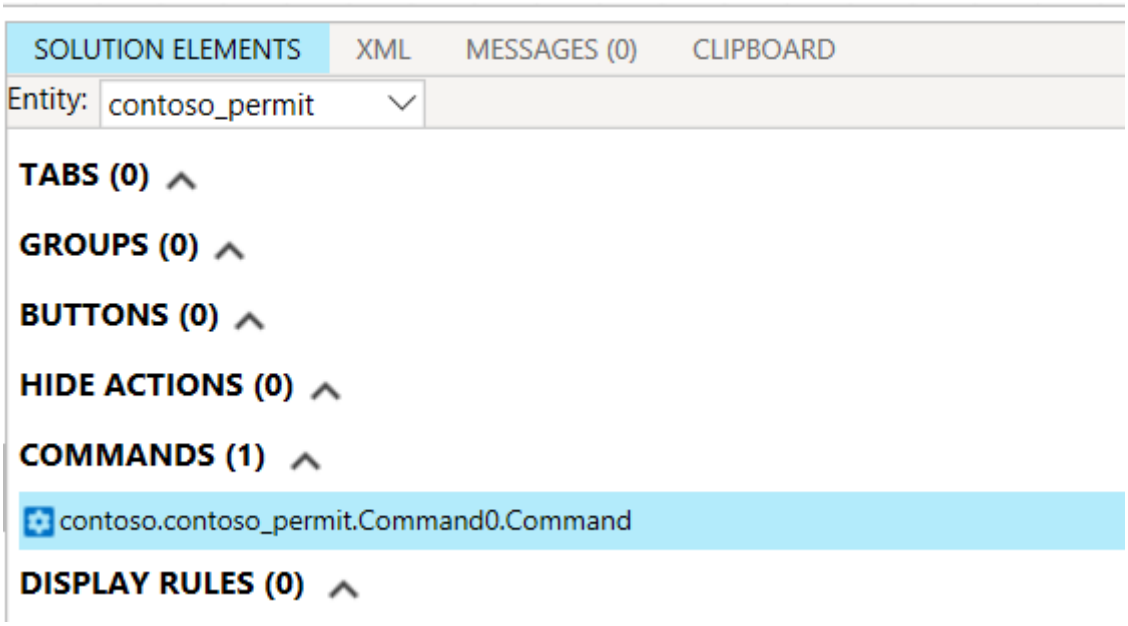


- Wait for the solution to be downloaded.

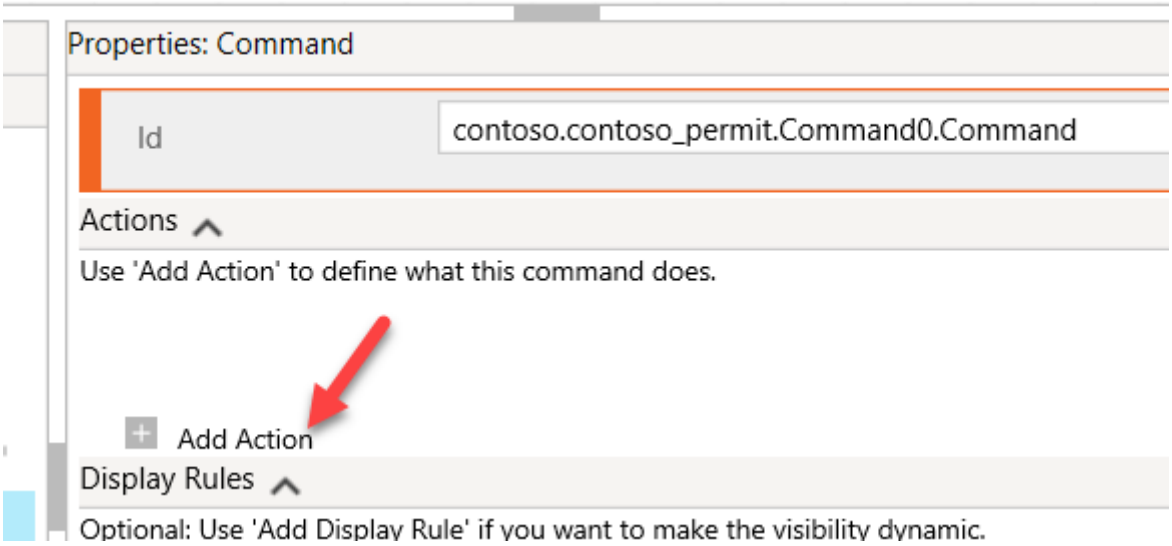
3. Add command to permit Table
   - Select **contoso_permit** from the dropdown for Table.
   - Click **Add Command**.



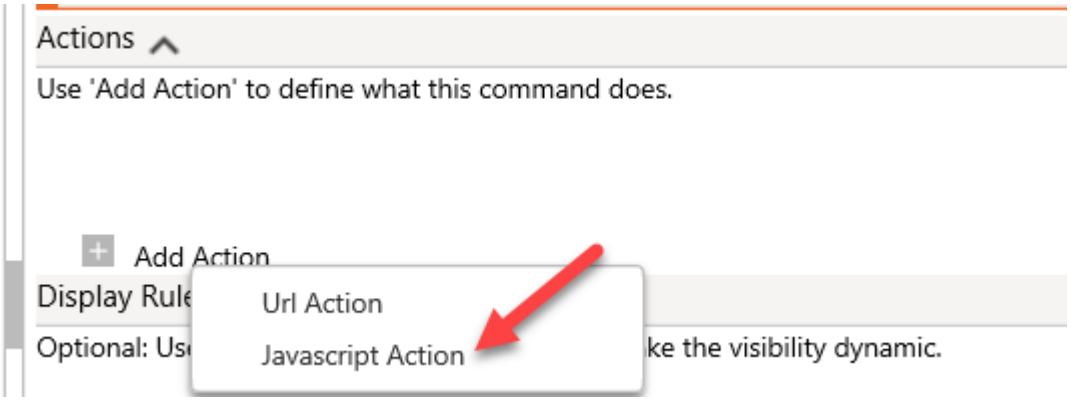
4. Add action to command
   - Select the command you just created.



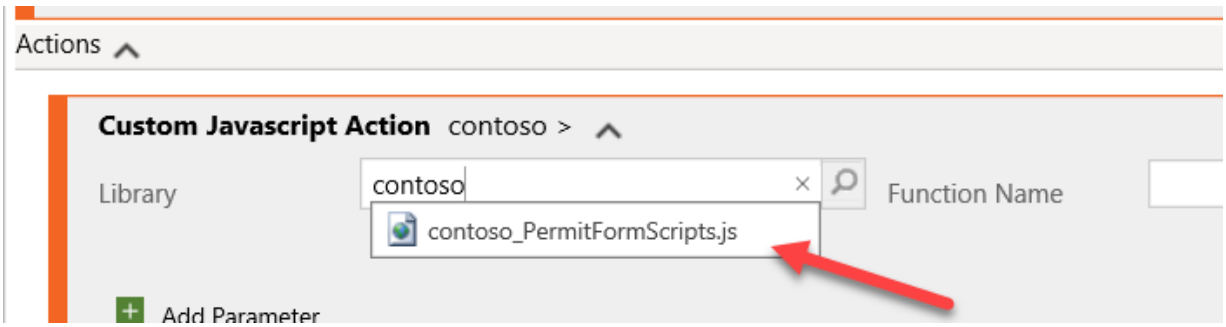
- Click **Add Action.**



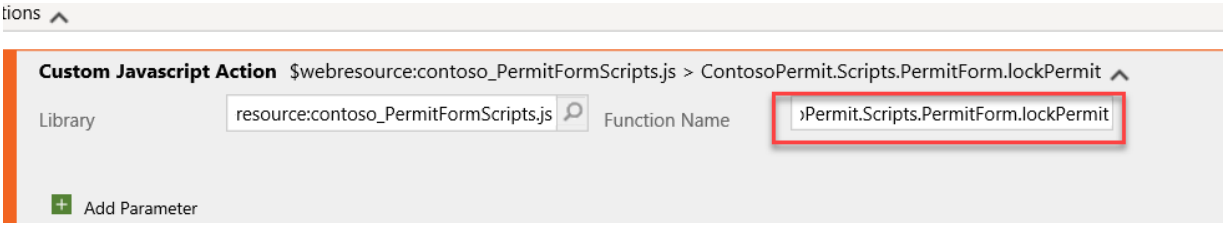
- Select **JavaScript Action**.



- Search for **contoso** and select **contoso_PermitFormScripts.js**

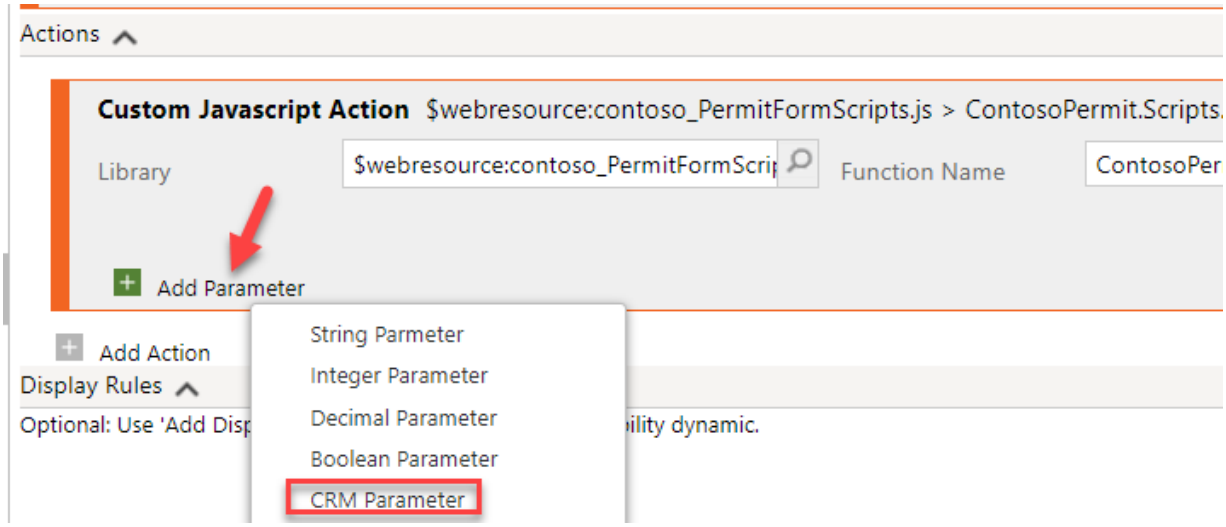


- Enter **ContosoPermit.Scripts.PermitForm.lockPermit** in the textbox for **Function Name**.

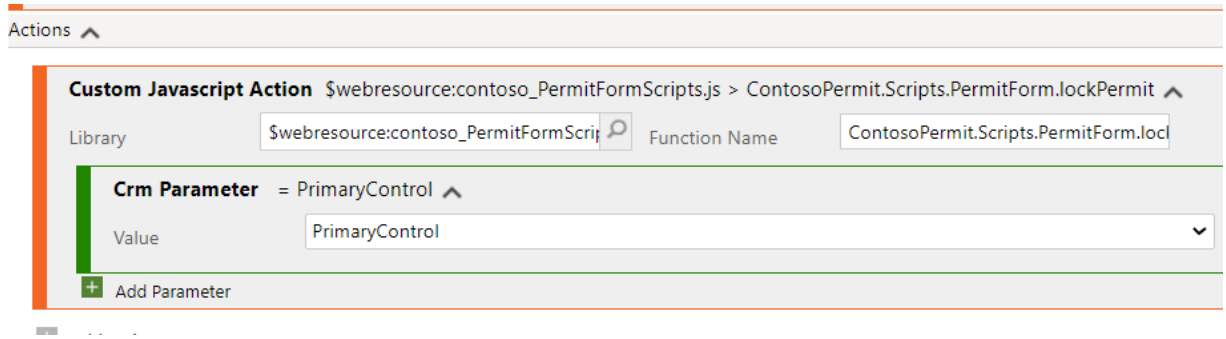


5. Add primary control parameter

- Click **Add Parameter** and select **CRM Parameter**.



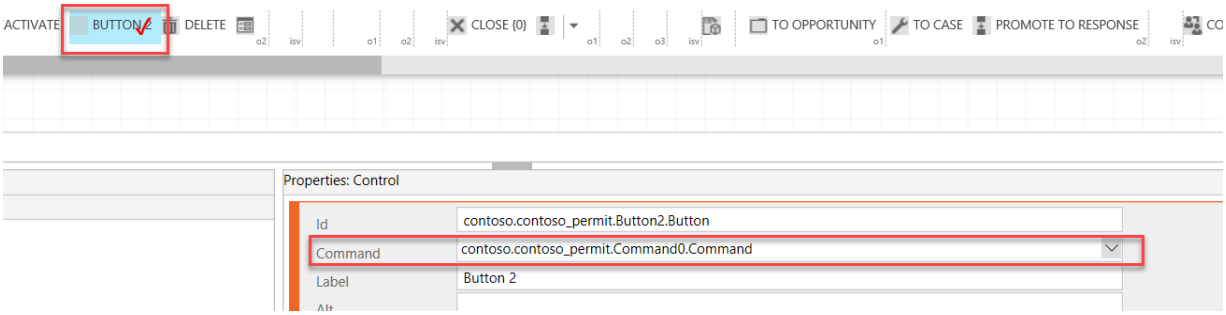
- Select **PrimaryControl** for **Value**.



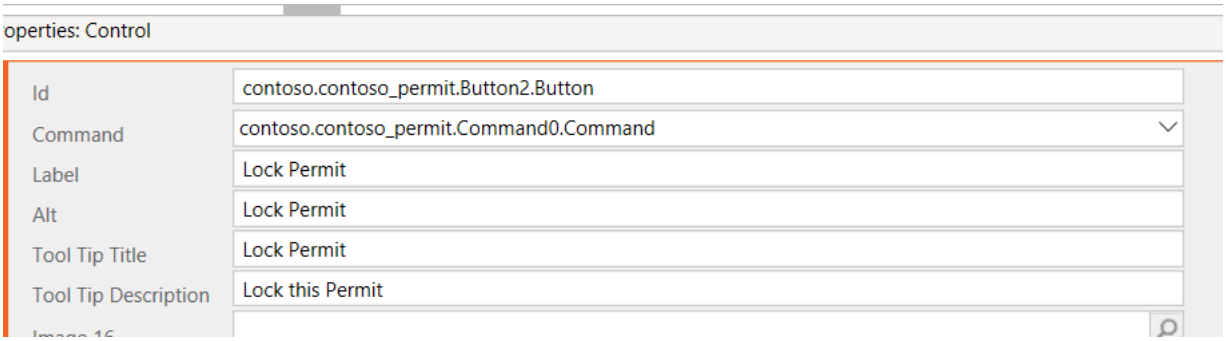
6. Add a button select command

- Drag a **Button** from the **Toolbox** and drop it between **Deactivate** and **Delete**. Make sure you are adding it on the Mscrm.Form.contoso_permit.MainTab section.

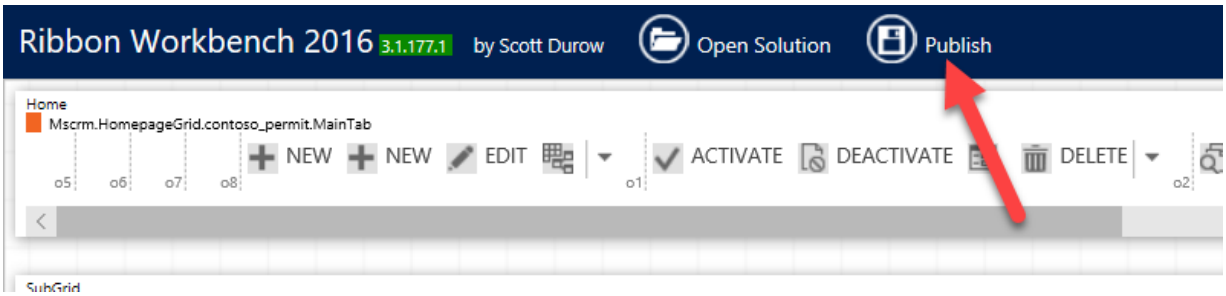
- Select the button you just added to ribbon.
- Go to the **Properties** section and select the command you created for **Command**.



- Change the **Label** value to **Lock Permit**.
- Provide **Alt** text, **Tool Tip Title**, and **Tool Tip Description.**



7. Publish your changes.



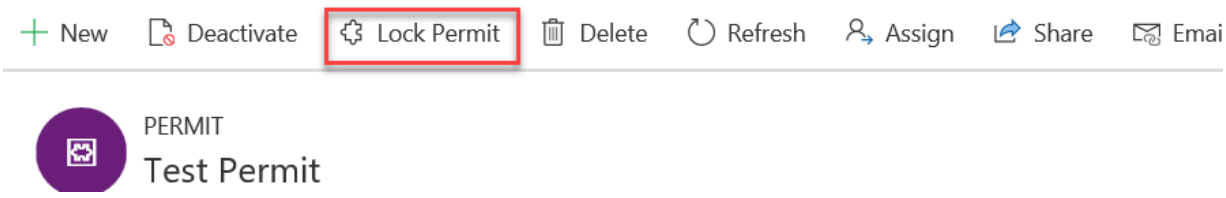
Click **OK** and wait for the publishing to complete. This might take a few minutes to complete.

21.5 Task #5: Test Command Button

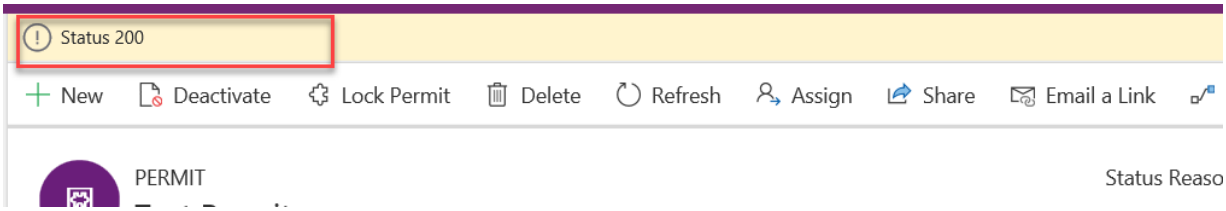
1. Start the Permit Management application.
   - Log on to Power Apps maker portal and make sure you have your **Dev** environment selected.

- Select **Apps** and click to open the **Permit Management** application.

2. Open a permit record
   - Select Permits.
   - Click to open a permit.
   - You should be able to see the button you just added.



3. Test the Command
   - Click on the **Lock Permit** button.
   - The script should trigger the action and you should be able to see the Status 200 notification.



21.6 lab: title: 'Lab 05: Power Apps Component Framework'

[!NOTE] Effective November 2020:

- Common Data Service has been renamed to Microsoft Dataverse. Learn more
- Some terminology in Microsoft Dataverse has been updated. For example, *entity* is now *table* and *field* is now *column*. Learn more

This content will be updated soon to reflect the latest terminology.

21.7 Lab 05 – Power Apps Component Framework

22 Scenario

A regional building department issues and tracks permits for new buildings and updates for remodeling of existing buildings. Throughout this course you will build applications and automation to enable the regional building department to manage the permitting process. This will be an end-to-end solution which will help you understand the overall process flow.

In this lab you will develop a custom component control using the Power Apps Component Framework (PCF). This component will be used to visualize the permit inspection history timeline. As you build the component, you will see how to use prescriptive interfaces of the framework to interact with the hosting form data. To speed things up we will use a community timeline library to render the visualization. When you build such controls, you can either follow the same procedure or use popular frameworks like React or Vue to completely control the visualization that the component will render.

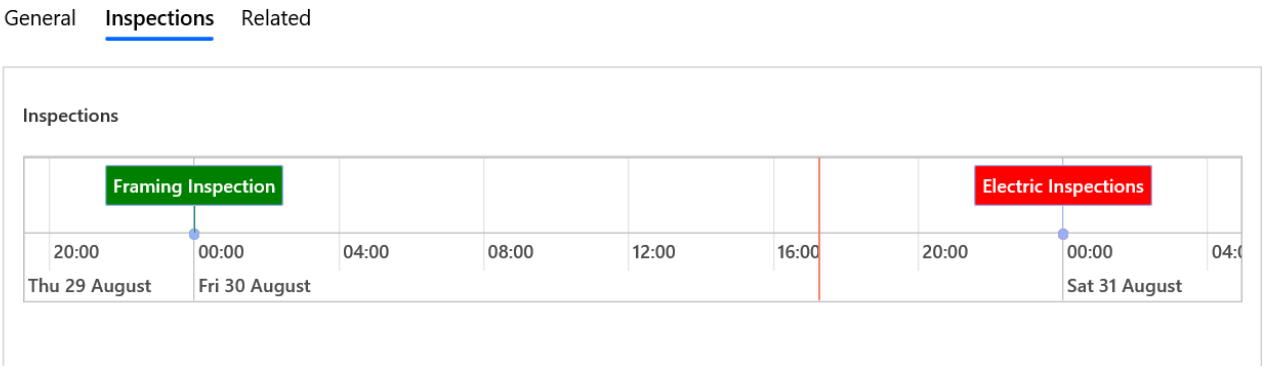
23 High-level lab steps

As part of building this component, you will complete the following steps:

- Use the Power Apps CLI to initialize the new component
- Build the component logic using Typescript
- Publish the component for use on forms

- Configure the permit form to use the component

The is what the component will look like when it is completed.



23.1 Things to consider before you begin

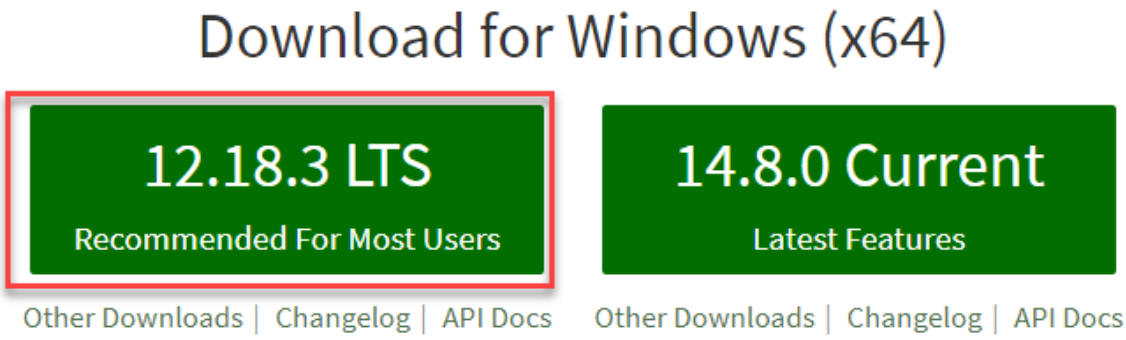
- What are the advantages of using a Power Apps Component Framework component over an embedded Power Apps canvas app?
- Remember to continue working in your DEVELOPMENT environment. We'll move everything to production soon.

24 Exercise #1: Create the PCF Control

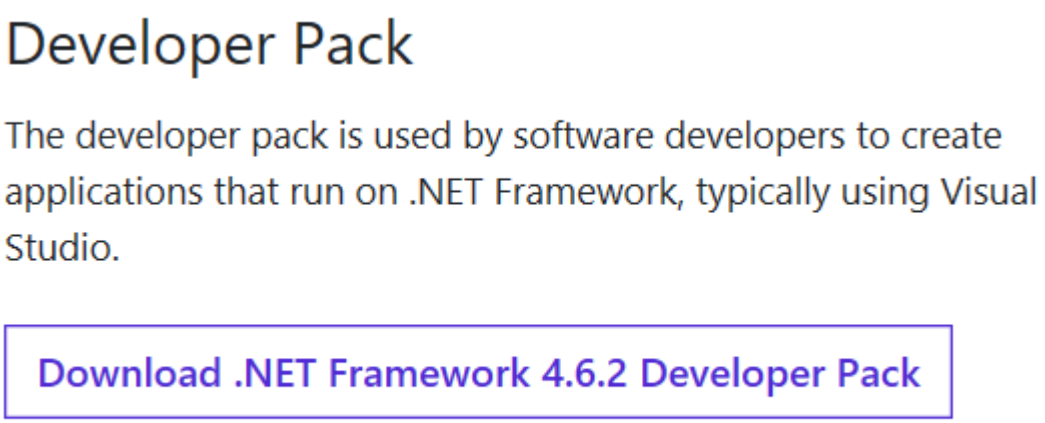
Objective: In this exercise, you will create a Power Apps Components Framework control using the Power Apps CLI

24.1 Task #1: Install Microsoft Power Apps CLI and Prerequisites

1. Install Node.js
   - Navigate to Node JS
   - Select the latest **LTS** version.



   - Click **Open file**.
   - Follow the steps in setup wizard to complete installing **Node.js**
2. Install .NET Framework 4.6.2 Developer Pack
   - Navigate to Download .NET Framework 4.6.2
   - Select the **Developer Pack**.

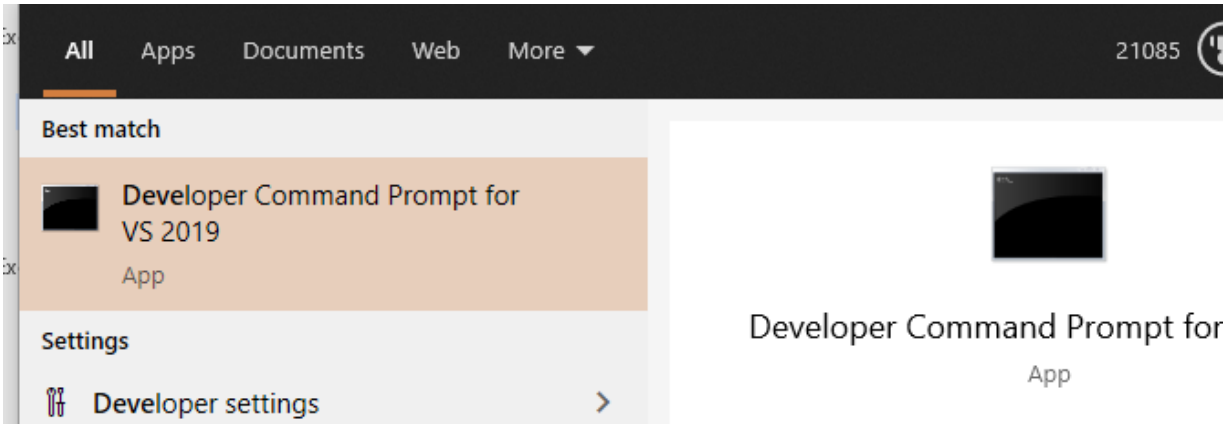


- Click **Open file**.

Follow the steps in setup wizard to complete installing the **Developer Pack.**

24.2 Task #2: Setup Components Project

1. Start the developer command prompt tool
   - Click **Start** and search for **developer**.
   - Click to start the **developer command prompt**.



2. Create a new folder in your Documents folder and work in that directory
   - Run the command mentioned below to change directory. Replace [**Computer User Name**] with your OS user name.

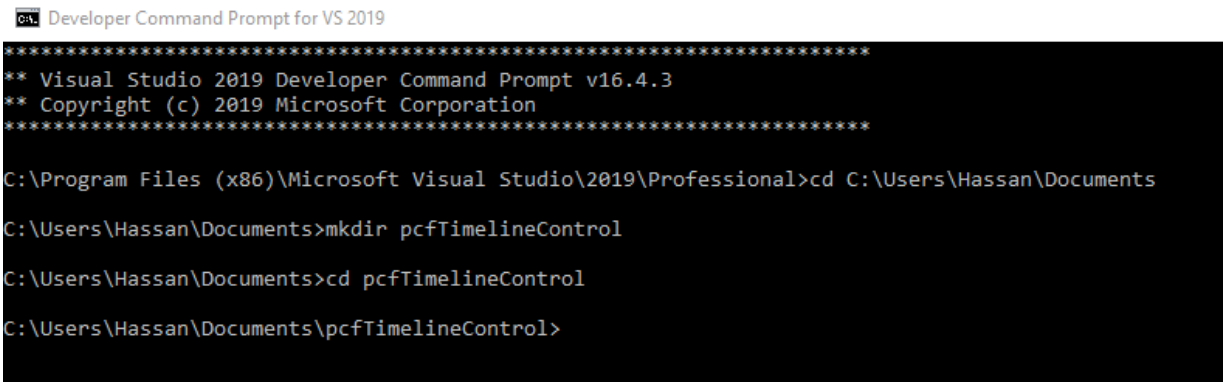
     ```
     cd C:\Users\[Computer User Name]\Documents
     ```

   - Run the command mentioned below to create a new folder with name **pcfTimelineControl**.

     ```
     mkdir pcfTimelineControl
     ```

   - Run the command mentioned below to switch to the **pcfTimelineControl** folder.

     ```
     cd pcfTimelineControl
     ```



3. Create a new folder in the **pcfTimelineControl** folder, name it **src**, and work in that directory
   - Create a new folder with the name **src**, by running the command below.

     ```
     mkdir src
     ```

   - Run the command below to switch to the **src** folder you just created.

     ```
     cd src
     ```

   - Clear the screen by running the command below.

     ```
     cls
     ```

4. Install the latest Power Apps CLI, create a solution project with the **name timelinecontrol**, **namespace contoso**, and **template dataset**.
   - Install latest **Power Apps CLI** version. Use: https://aka.ms/PowerAppsCLI
     Note: if you just installed the tools, you already have the latest, however, you can run this command anytime to ensure you are always up to date.

     ```
     pac install latest
     ```

   - Initialize the component. This command will create a set of files that will implement a dataset component. You will customize these files as per your specific component as we continue.

     ```
     pac pcf init --name timelinecontrol --namespace contoso --template dataset
     ```

   - Install dependencies by running **npm install** command in the Terminal

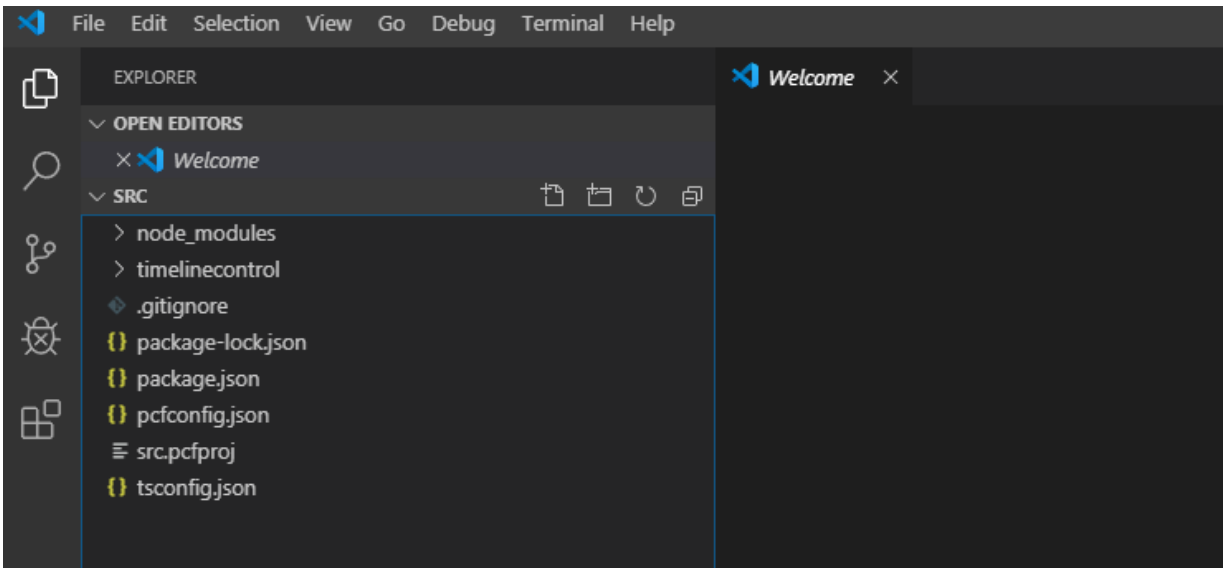
     ```
     npm install
     ```

   - Wait for the dependency installation to complete.

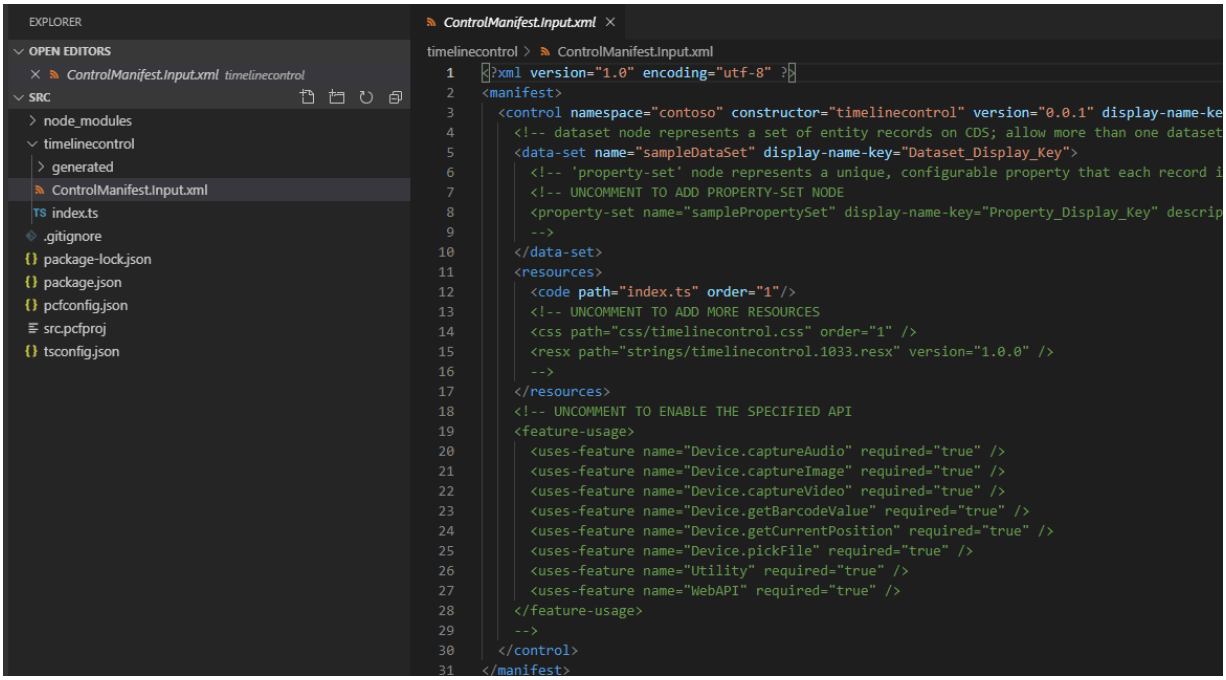
5. Open the **src** folder in Visual Studio Code and review the generated resources.
   - Open the **src** folder in **Visual Studio Code**. For this to work, make sure that the Visual Studio Code is added to Path in Environment Variables.

     ```
     code .
     ```

   - **Visual Studio Code** should start, and it should open the **src** folder.

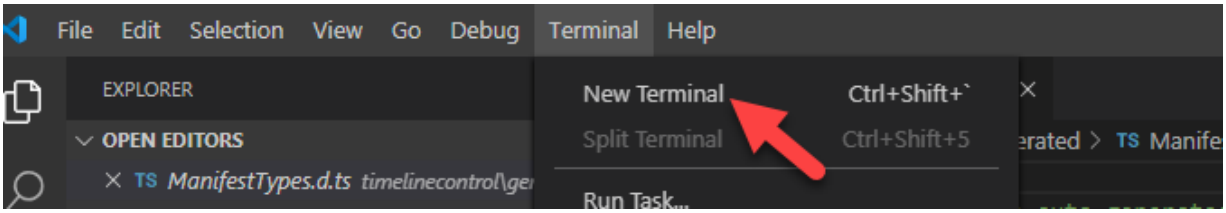


- Expand the **timelinecontrol** folder.
- Open the **ControlManifest.Input** xml file and examine it.

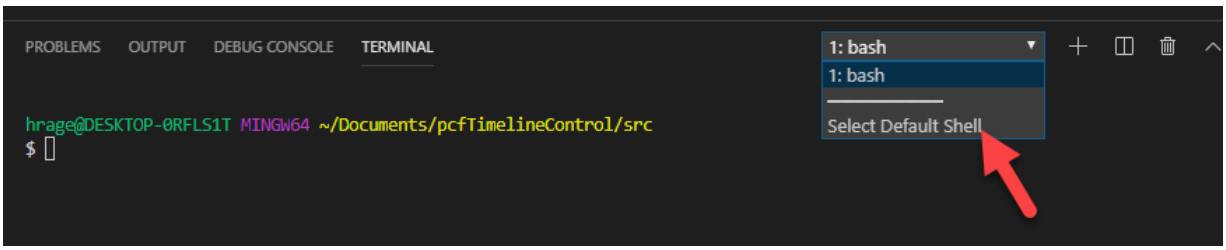


- Open the **Index.ts** file and examine it.
- Expand the **generated** folder.
- Open the **ManifestTypes** file and examine it.

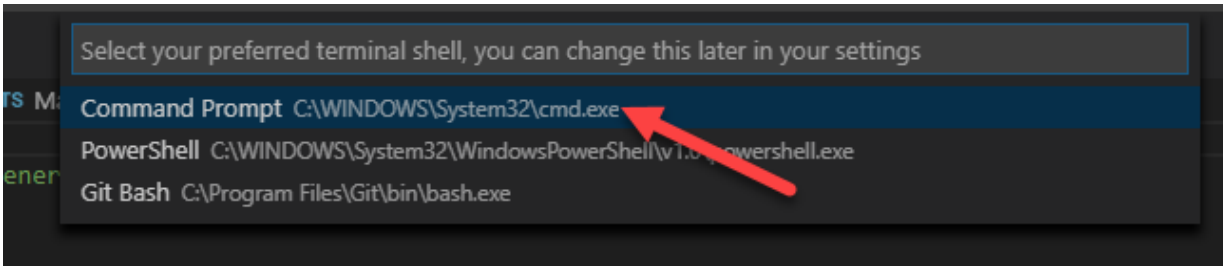
6. Open CLI in visual studio code.
   - Click **Terminal** and select **New Terminal**. If Terminal is not visible in the Top menu, you can open it by selecting View -> Integrated Terminal.



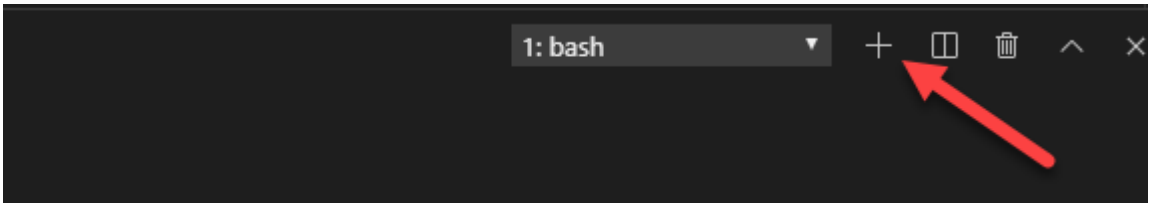
- If **cmd** isn't your **Default Shell** open, click on the arrow and click **Select Default Shell**.



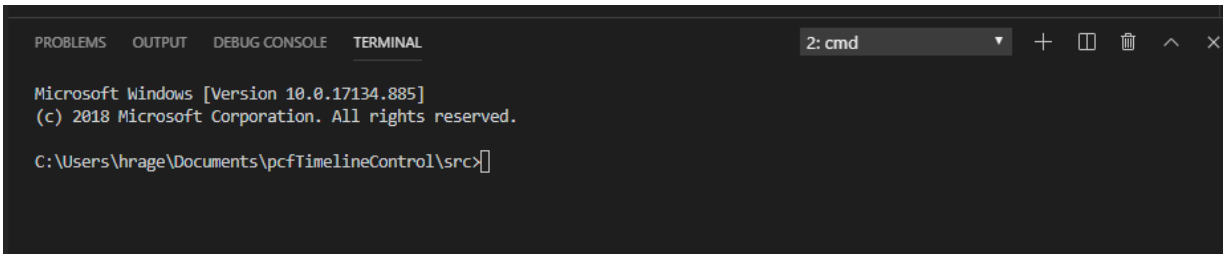
- Select **Command Prompt**.



- Click **New Terminal**.



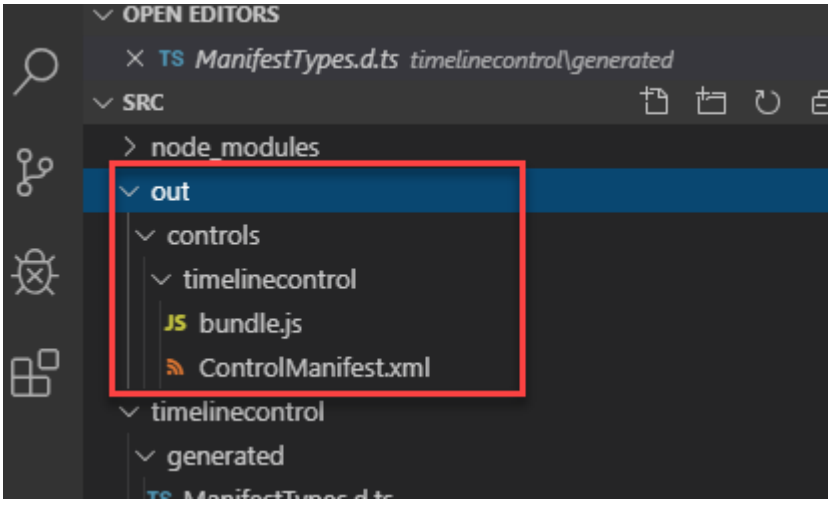
- The **cmd** terminal should now open.



7. Run the Build command and review the out folder.
   - Run **npm build** in the terminal

     ```
     npm run build
     ```

   - You should now be able to see the out folder. Expand the out folder and review its content.

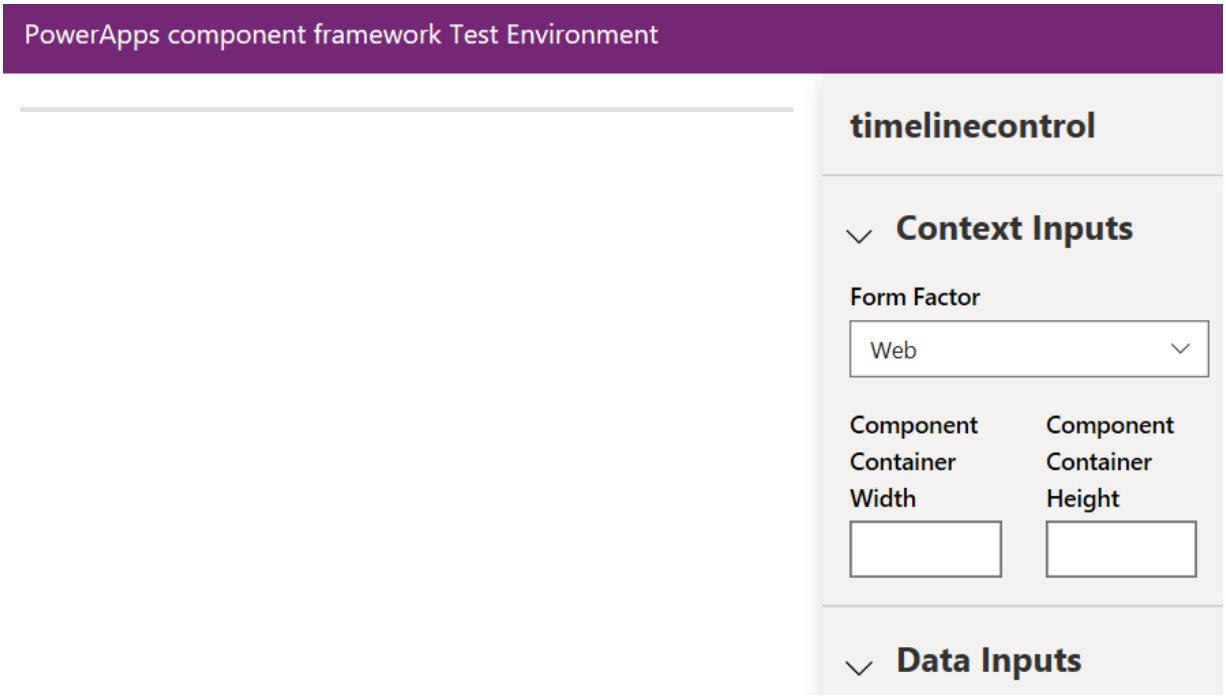


8. Run the Start command to start the test harness.

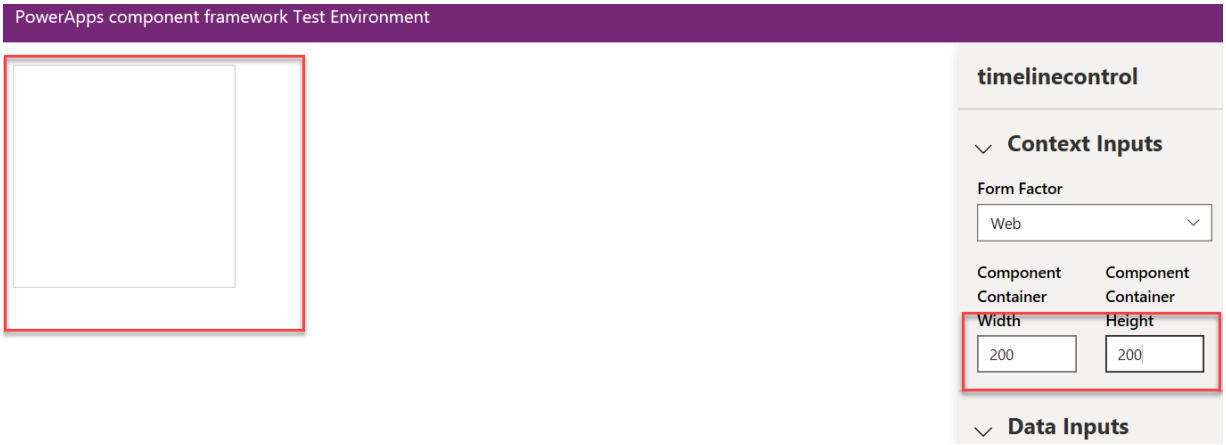
- Run **npm start** in the terminal

```
npm start
```

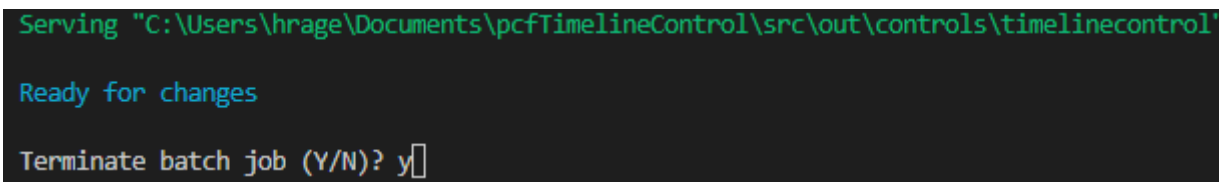
- This should open the Test Environment in a browser window.



- The **Component** container size should change, if you provide **Width** and **Height**.



9. Stop the test harness
   - Close the **Test Environment** browser window or tab.
   - Go back to **Visual Studio Code**.
   - Click on the **Terminal** and press the [**CONTROL**] key and **c**.
   - Type **y** and [**ENTER**].



10. Create a new solution folder in the parent of the **src** folder **pcfTimelineControl** and switch to it.
    - Change directory to the **pcfTimelineControl** folder.

```
cd ..
```

- You should now be in the **pcfTimelineControl** directory.

```
Terminate batch job (Y/N)? y

C:\Users\hrage\Documents\pcfTimelineControl\src>cd ..

C:\Users\hrage\Documents\pcfTimelineControl>
```

- Create a new folder with the name **solution**.

```
mkdir solution
```

- Switch to the solution directory.

```
cd solution
```

- You should now be in the solution directory.

```
C:\Users\hrage\Documents\pcfTimelineControl>cd solution

C:\Users\hrage\Documents\pcfTimelineControl\solution>
```

11. Create solution project and add reference of the **src** folder where the component is located to the solution. This configuration will be used when you are done with your development and ready to publish your component for others to use.

- Create solution project with name and prefix contoso.

```
pac solution init --publisher-name contoso --publisher-prefix contoso
```

- Add component location to the solution. This creates a reference to include your component when the solution build command is run.

```
pac solution add-reference --path ..\src
```

- The project reference should be added successfully.

```
C:\Users\hrage\Documents\pcfTimelineControl\solution>pac solution  add-reference --path ..\src
Project reference successfully added to CDS solution project.

C:\Users\hrage\Documents\pcfTimelineControl\solution>
```

12. Build the solution

- Make sure you are still in the solution folder.
- Build the project by running the command below.

```
msbuild /t:restore
```

- The build should succeed.

```
Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:04.52

C:\Users\hrage\Documents\pcfTimelineControl\solution>
```

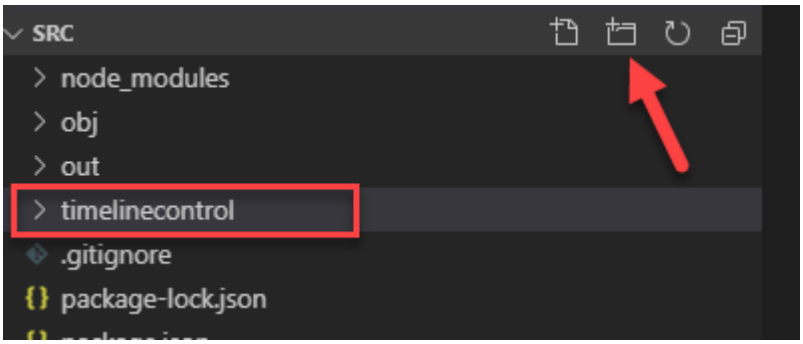
24.3 Task #3: Build the Basic Timeline

1. Change directory to the **src** folder

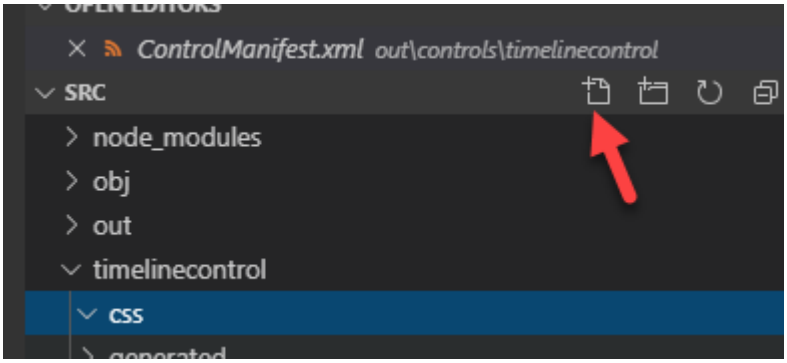
- Change directory to the **src** folder.

```
cd ..\src
```

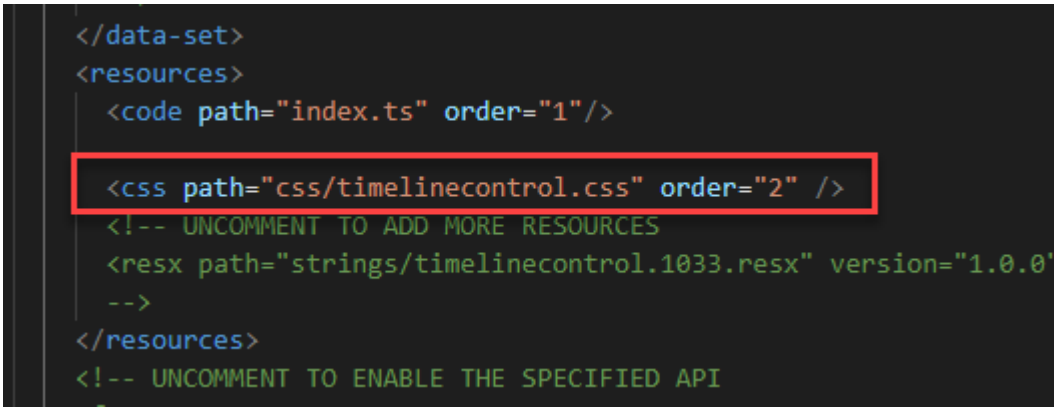
2. Create **css** folder in the **timelinecontrol** folder and create **timelinecontrol.**css file in the **css** folder
   - Select the **timelinecoltrol** folder and click **New Folder**.



   - Name the folder **css**.
   - Select the **css** folder you created and click **New File**.



   - Name the new file **timelinecontrol.css**.
3. Add the **css** as resource
   - Open the **ControlManifest.Input.xml** file.
   - Locate the **resources** sub element and uncomment the **css** tag, change the **Order** to **2**.



4. Change the data-set name to **timelineDataSet**.
   - Locate **data-set** tag and change the name property to **timelineDataSet**.

```
<manifest>
  <control namespace="contoso" constructor="timelinecontrol" version="0.0.1" d
    <!-- dataset node represents a set of entity records on CDS; allow more th
    <data-set name="timelineDataSet" display-name-key="Dataset_Display_Key">
      <!-- 'property-set' node represents a unique, configurable property that
      <!-- UNCOMMENT TO ADD PROPERTY-SET NODE
      <property-set name="samplePropertySet" display-name-key="Property_Displa
      -->
```

5. Install vis-timeline css npm package
   - Go to the **Terminal** and make sure you are in the **src** directory.
   - Run the command mentioned below and wait for the packages to be added.

     ```
     npm install vis-timeline
     ```

```
+ vis-timeline@5.1.0
added 9 packages from 139 contributors and audited 10285 packages in 8.131s
found 0 vulnerabilities

C:\Users\hrage\Documents\pcfTimelineControl\src>
```

   - Run the command mentioned below and wait for the packages to be added.

     ```
     npm install moment
     ```

   - Run the command mentioned below and wait for the packages to be added.

     ```
     npm install vis-data
     ```

6. Add the vis-timeline css as a resource
   - Go back to the **ControlManifest.Input.xml** file.
   - Add the vis-timeline css inside the resources tag.

     ```
     <css path="..\node_modules\vis-timeline\dist\vis-timeline-graph2d.min.css" order="1" />
     ```

```
<resources>
  <code path="index.ts" order="1"/>
  <css path="..\node_modules\vis-timeline\dist\vis-timeline-graph2d.min.css" order="1" />
  <css path= css/timelinecontrol.css  order= 2  />
  <!-- UNCOMMENT TO ADD MORE RESOURCES
  <resx path="strings/timelinecontrol.1033.resx" version="1.0.0" />
  -->
</resources>
<!-- UNCOMMENT TO ENABLE THE SPECIFIED API
```

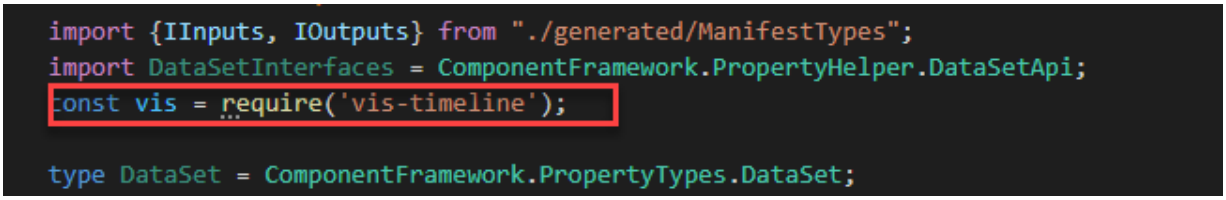
7. Add timeline element and visual properties to the Index file
   - Open the **Index.ts** file.
   - Add the properties below, inside the **export** class timelinecontrol function.

     ```
     private _timelineElm: HTMLDivElement;
     private _timelineVis : any;
     ```

```
5   export class timelinecontrol implements ComponentFramework.StandardControl<IInputs, IOutputs> {
6
7       private _timelineElm: HTMLDivElement;
8       private _timelineVis : any;
9
10      /**
11       * Empty constructor.
12       */
13      constructor()
14      {
```

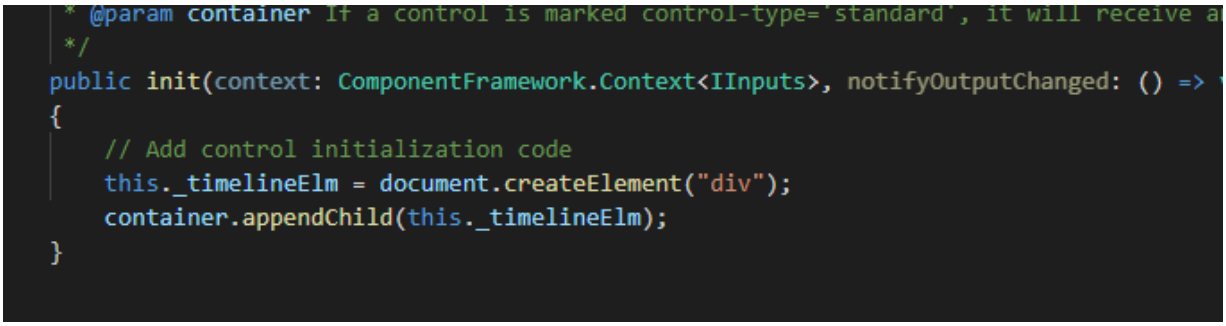
- Add the below constant after the import lines on the top.

```
const vis = require('vis-timeline');
```



8. Build the timeline element as div and add it to container element as a child.
   - Locate the **init** method.
   - Add the script mentioned below to the **init** function.

```
this._timelineElm = document.createElement("div");

container.appendChild(this._timelineElm);
```



9. Create a function that will render the timeline
   - Add the function below.

```
private renderTimeline(): void {
    // Create a DataSet (allows two way data-binding)
    var items = [
        { id: 1, content: 'item 1', start: '2020-08-20' },
        { id: 2, content: 'item 2', start: '2020-08-14' },
        { id: 3, content: 'item 3', start: '2020-08-18' },
        { id: 4, content: 'item 4', start: '2020-08-16', end: '2020-08-19' },
        { id: 5, content: 'item 5', start: '2020-08-25' },
        { id: 6, content: 'item 6', start: '2020-08-27', type: 'point' }
    ];
    // Configuration for the Timeline
    var options = {};
    // Create a Timeline
    var timeline = new vis.Timeline(this._timelineElm, items, options);
}
```

```
private renderTimeline(): void {
    // Create a DataSet (allows two way data-binding)
    var items = [
        { id: 1, content: 'item 1', start: '2020-08-20' },
        { id: 2, content: 'item 2', start: '2020-08-14' },
        { id: 3, content: 'item 3', start: '2020-08-18' },
        { id: 4, content: 'item 4', start: '2020-08-16', end: '2020-08-19' },
        { id: 5, content: 'item 5', start: '2020-08-25' },
        { id: 6, content: 'item 6', start: '2020-08-27', type: 'point' }
    ];
    // Configuration for the Timeline
    var options = {};
    // Create a Timeline
    var timeline = new vis.Timeline(this._timelineElm, items, options);
}
```

10. Call the **renderTimeline** function from the **updateView** function.
    - Locate the **updateView** function.
    - Add the script mentioned below inside the **updateView** function.

      ```
      this.renderTimeline();
      ```

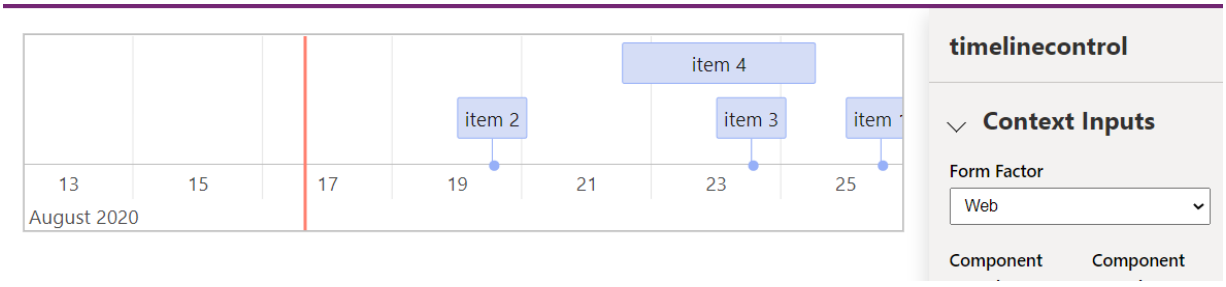
```
    */
    public updateView(context: ComponentFramework.Context<IInputs>): void {
        // Add code to update control view
        this.renderTimeline();
    }

    private renderTimeline(): void {
```

    - Click **File** and **Save All**.
11. Build and start
    - Go to the **Terminal** and make sure you are still in the **src** directory.
    - Run the build command.

      ```
      npm run build
      ```

    - The build should succeed.
    - Run the start watch command. This command will keep the test environment running and auto update when you change the component.

      ```
      npm start watch
      ```

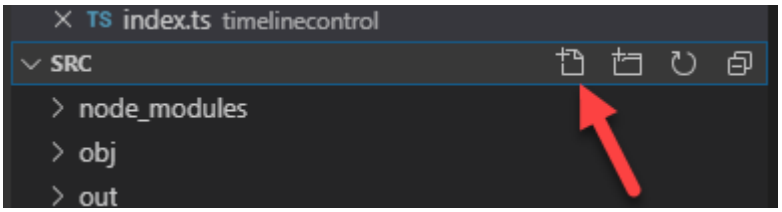


Do not close the test environment.

24.4 Task #4: Tailor for Inspection Data

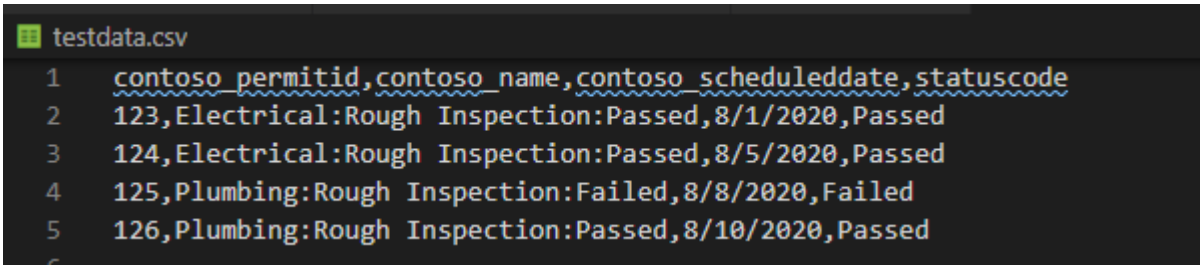
In this task, you will switch from using the hard-coded array of data to using a file loaded into the test harness.

1. Create test data csv file
   - Select the **src** folder. And click **New File**.



   - Name the new file **testdata.csv**
   - Add the below mentioned data inside the **testdata.csv** file and Save it.

```
contoso_permitid,contoso_name,contoso_scheduleddate,statuscode
123,Electrical:Rough Inspection:Passed,8/1/2020,Passed
124,Electrical:Rough Inspection:Passed,8/5/2020,Passed
125,Plumbing:Rough Inspection:Failed,8/8/2020,Failed
126,Plumbing:Rough Inspection:Passed,8/10/2020,Passed
```



2. Create Timeline Data class
   - Open the **index.ts** file.
   - Paste the code below after the **type DataSet** line.

```
class TimelineData {
    id: string;
    content: string;
    start: string;
    className: string;

    constructor(id: string, content: string, start: string, className: string) {
    this.id = id;
    this.content = content;
    this.start = start;
    this.className = className;
    }
}
```

```
type DataSet = ComponentFramework.PropertyTypes.DataSet;

class TimelineData {
    id: string;
    content: string;
    start: string;
    className: string;

    constructor(id: string, content: string, start: string, className: string) {
        this.id = id;
        this.content = content;
        this.start = start;
        this.className = className;
    }
}

export class timelinecontrol implements ComponentFramework.StandardControl<IInputs, IOutputs> {
```

- Add the timeline data array property inside the **export** class timelinecontrol function and below the **_timelineElm** definition.

```
private _timelineData : TimelineData[] = [];
```

```
20
21  export class timelinecontrol implements ComponentFramework.StandardControl<IInputs, IOutputs> {
22
23      private _timelineElm: HTMLDivElement;
24      private _timelineData : TimelineData[] = [];
25      private _timelineVis: any;
26
27      /**
28       * Empty constructor.
```

3. Add a method that will create the timeline data.

- Add the method mentioned below after the **render** method.

```
private createTimelineData(gridParam: DataSet) {
    this._timelineData = [];
    if (gridParam.sortedRecordIds.length > 0) {
        for (let currentRecordId of gridParam.sortedRecordIds) {

            console.log('record: ' + gridParam.records[currentRecordId].getRecordId());

            var permitName = gridParam.records[currentRecordId].getFormattedValue('contoso_n
            var permitDate = gridParam.records[currentRecordId].getFormattedValue('contoso_s
            var permitStatus = gridParam.records[currentRecordId].getFormattedValue('statusc
            var permitColor = "green";
            if (permitStatus == "Failed")
                permitColor = "red";
            else if (permitStatus == "Canceled")
                permitColor = "yellow";


            console.log('name:' + permitName + ' date:' + permitDate);


            if (permitName != null)
                this._timelineData.push(new TimelineData(currentRecordId, permitName, permit
        }
    }
    else {
        //handle no data
    }
```

```
    }
```

```
}

private createTimelineData(gridParam: DataSet) {
    this._timelineData = [];
    if (gridParam.sortedRecordIds.length > 0) {
        for (let currentRecordId of gridParam.sortedRecordIds) {

            console.log('record: ' + gridParam.records[currentRecordId].getRecordId());

            var permitName = gridParam.records[currentRecordId].getFormattedValue('contoso_name')
            var permitDate = gridParam.records[currentRecordId].getFormattedValue('contoso_scheduleddate')
            var permitStatus = gridParam.records[currentRecordId].getFormattedValue('statuscode')
            var permitColor = "green";
            if (permitStatus == "Failed")
                permitColor = "red";
            else if (permitStatus == "Canceled")
                permitColor = "yellow";

            console.log('name:' + permitName + ' date:' + permitDate);

            if (permitName != null)
                this._timelineData.push(new TimelineData(currentRecordId, permitName, permitDate, permitColor));
        }
    }
    else {
        //handle no data
    }
}
```

4. Call the createTimelineData method from the updateView method.
   - Go to the **updateView** method.
   - Replace the code inside the **updateView** method with the code below.

```
if (!context.parameters.timelineDataSet.loading) {
    // Get sorted columns on View
    this.createTimelineData(context.parameters.timelineDataSet);
    this.renderTimeline();
}
```

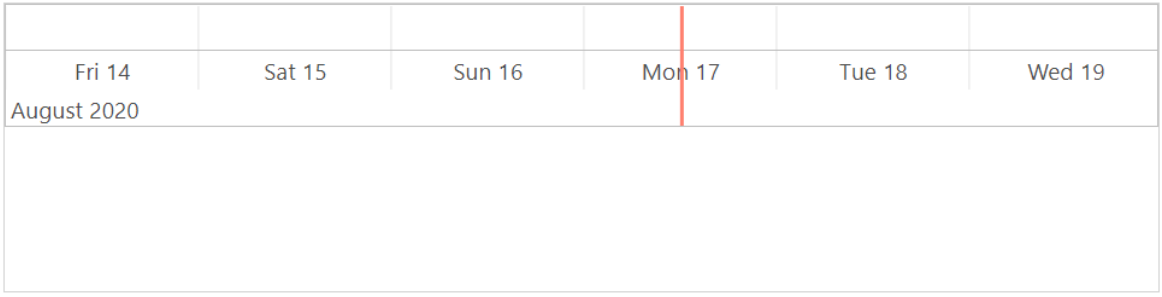
```
public updateView(context: ComponentFramework.Context<IInputs>): void {
    // Add code to update control view
    if (!context.parameters.timelineDataSet.loading) {
        // Get sorted columns on View
        this.createTimelineData(context.parameters.timelineDataSet);
        this.renderTimeline();
    }
}
```

5. Replace the hardcoded items with the csv data.
   - Locate the **renderTimeline** method.
   - Replace the hardcoded **items** with code below.

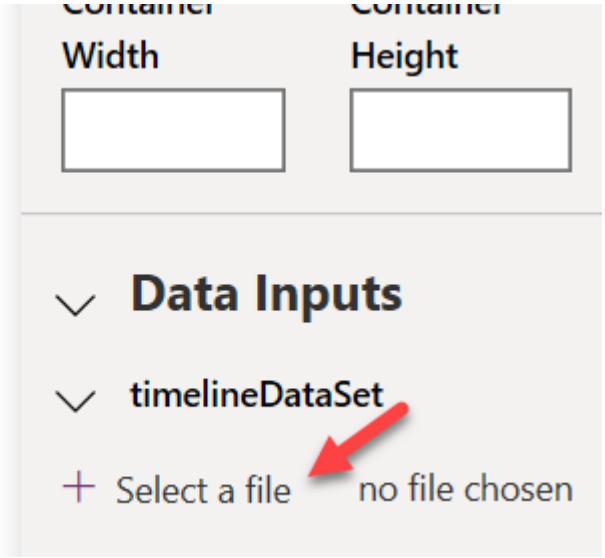
```
var items = this._timelineData;
```

```
private renderTimeline(): void {
    // Create a DataSet (allows two way data-binding)
    var items = this._timelineData;
    // Configuration for the Timeline
    var options = {};
    // Create a Timeline
    var timeline = new vis.Timeline(this._timelineElm, items,  options);
}
```

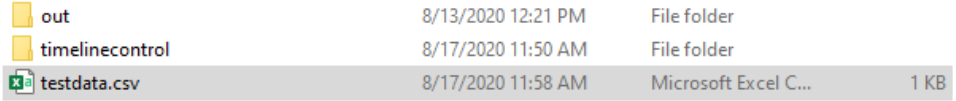
6. Make sure the test environment shows your changes and test the timeline control with the test data.
   - Click **File** and then **Save All**.
   - The test harness should still be running. If it is not running run **npm start watch** command.
   - Go to the test environment and make sure it looks like the image below.



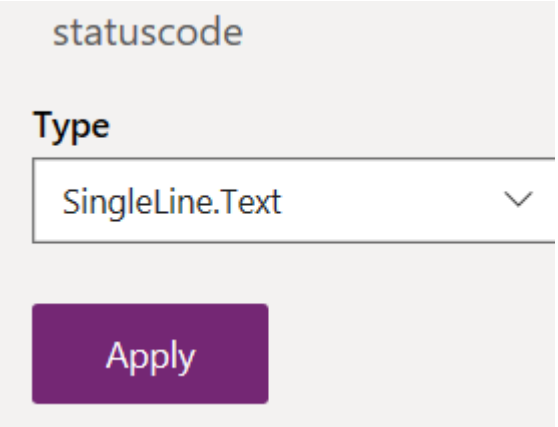
   - Click **Select a File**.



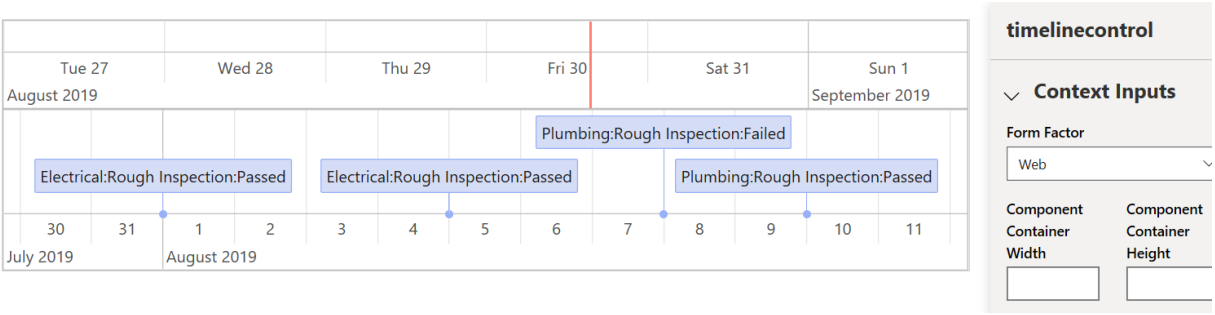
   - Select the **testdata.csv** and click **Open**.



   - Click **Apply**.



- The timeline control should now show the test data.



Do not close the test environment.

24.5 Task #5: Change Color for Items

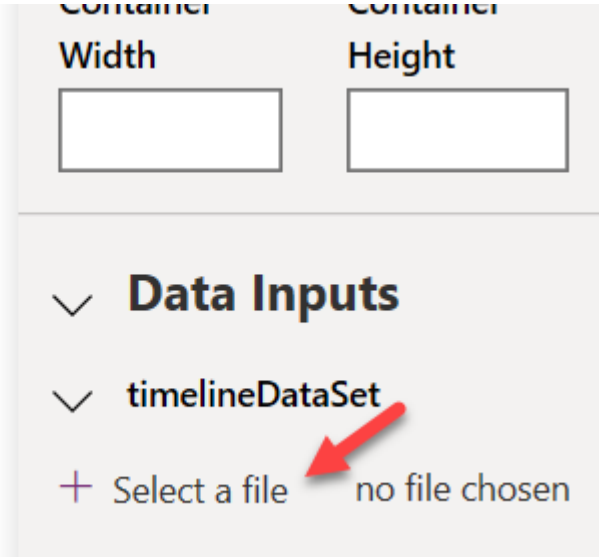
In this task, you will use the **css** resource you configured to change the color of the items on the timeline.

1. Add red and green styles to the timelinecontrol.css file
   - Go back to **Visual Studio Code**.
   - Expand the **css** folder and open the **timelinecontrol.css**
   - Add the style below to the **timelinecontrol.css** file and save your changes.

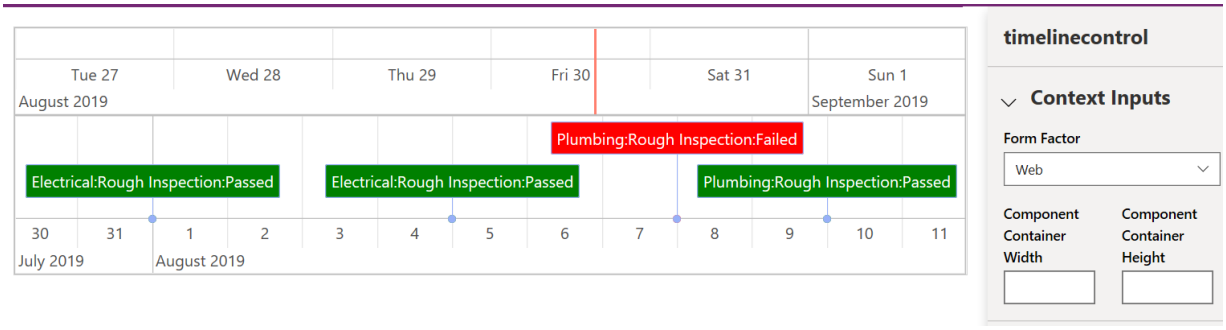
```
.red{
    background:red;
    color:white;
    }
.green{
    background:green;
    color:white;
    }
.yellow{
    background:yellow;
    color:black;
    }
```

```
# timelinecontrol.css ×   TS index.ts
src > timelinecontrol > css > # timelinecontrol.css > ...
1   .red{
2       background: red;
3       color: white;
4       }
5   .green{
6       background: green;
7       color: white;
8       }
9   .yellow{
10      background: yellow;
11      color: black;
12  }
```

2. Check the test environment, load the test data and make sure it shows your changes
   - Go to the **Test Environment**.
   - Click **Select a File**.



   - Select the **testdata.csv** and click **Open**.
   - Click **Apply**.
   - The timeline control should now show the test data.



   - Close the test environment browser window or tab.
3. Stop the test
   - Go back to **Visual Studio Code**.

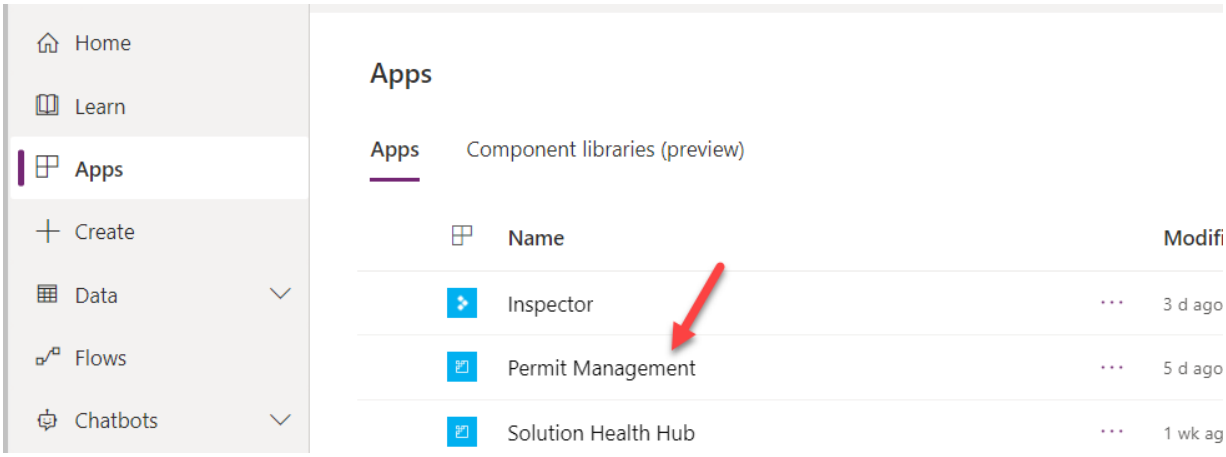
- Click on the **Terminal** and press the [**CONTROL**] key and **c**.
- Type **y** and [**ENTER**].

25 Exercise #2: Publish to Microsoft Dataverse

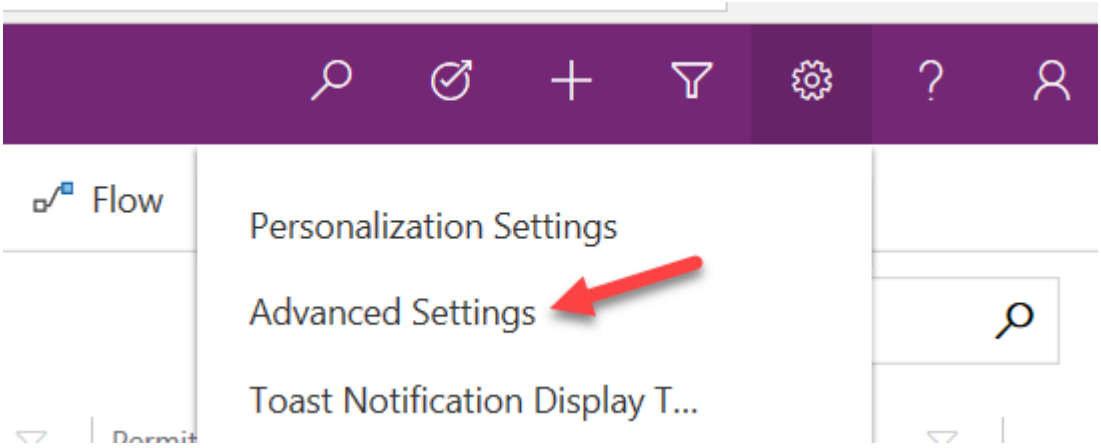
Objective: In this exercise, you will publish the timeline control to your Microsoft Dataverse and add it to the Permit main form.

25.1 Task #1: Setup and Publish

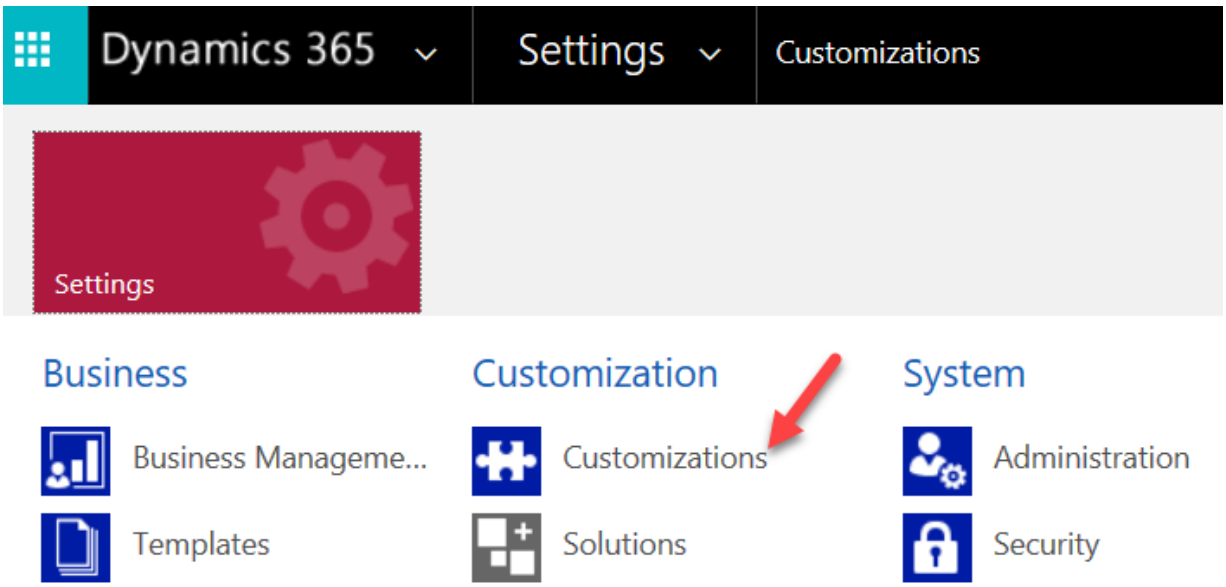
1. Get your Microsoft Dataverse org URL
   - Navigate to Power Apps maker portal and make sure you have the **Dev** environment selected.
   - Select **Apps** and open the Permit Management application.



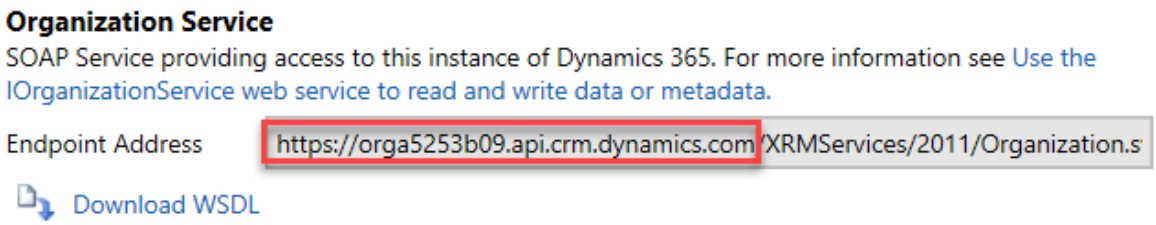
   - Click Settings and select Advanced Settings.



   - Navigate to **Settings** | **Customizations**.



- Click **Developer Resources**.
- Copy your organization **URL**.

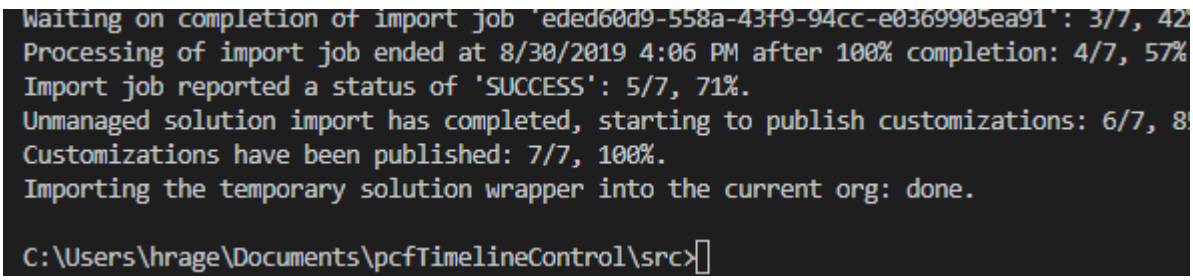


2. Authenticate
   - Go back to **Visual Studio Code**.
   - Make sure you are still in the **src** directory.
   - Run the command below. Replace **<orgurl>** with **URL** you copied.

     ```
     pac auth create --url <orgurl>
     ```

   - Sign in with your **admin** username.
3. Import the solution into your org and publish
   - Run the command below and wait for the publishing to complete. The push command uploads your component to the configured environment. This can be used over and over during development to quickly see your component in the live form.

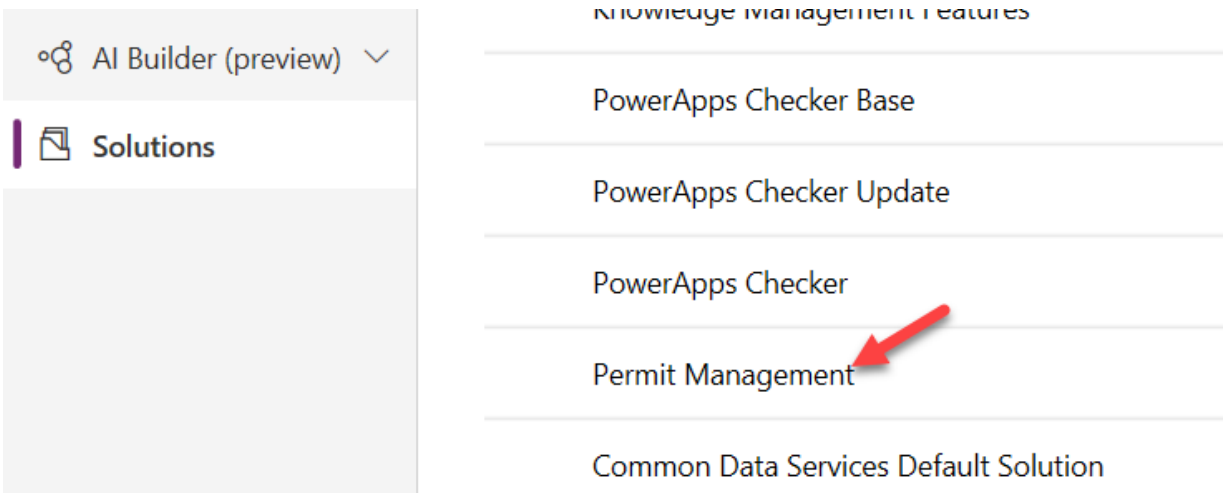
     ```
     pac pcf push --publisher-prefix contoso
     ```



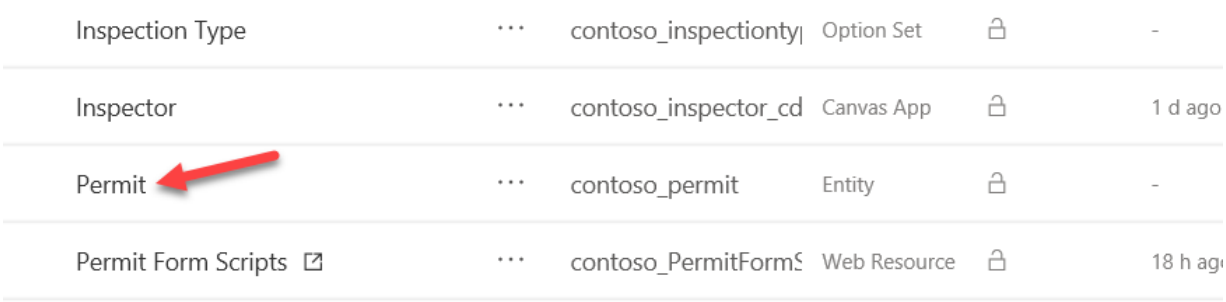
25.2 Task #2: Add Timeline Control to the Permit Form

1. Open the Permit Management solution.
   - Navigate to Power Apps maker portal and make sure you have the **Dev** environment selected.

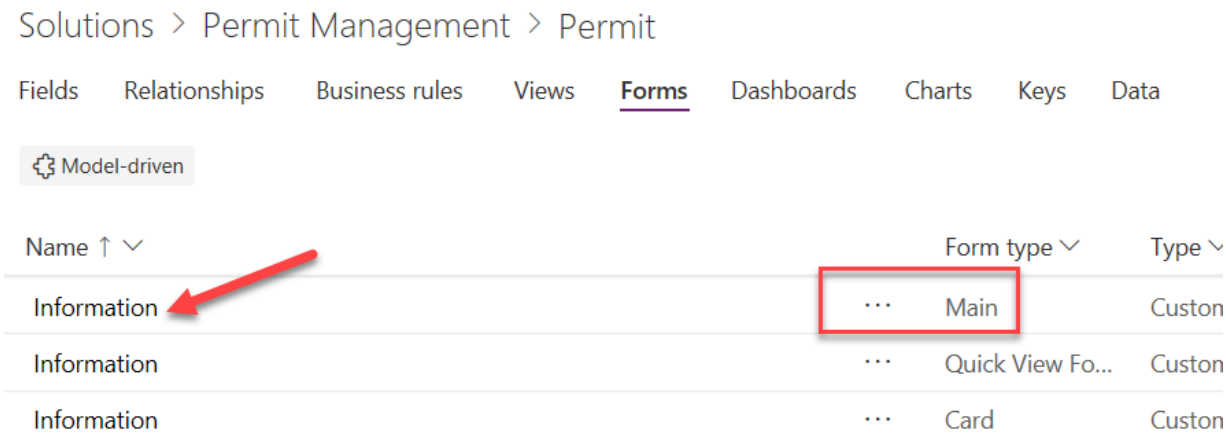
- Select **Solutions**.
- Click to open the **Permit Management** solution.



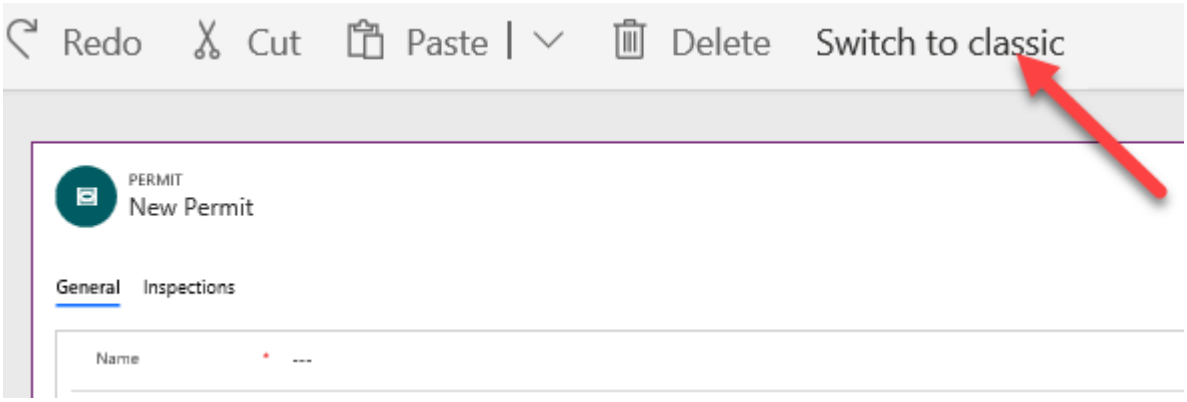
2. Open the Permit Main form and switch to classic
   - Locate and click to open the **Permit** Table.



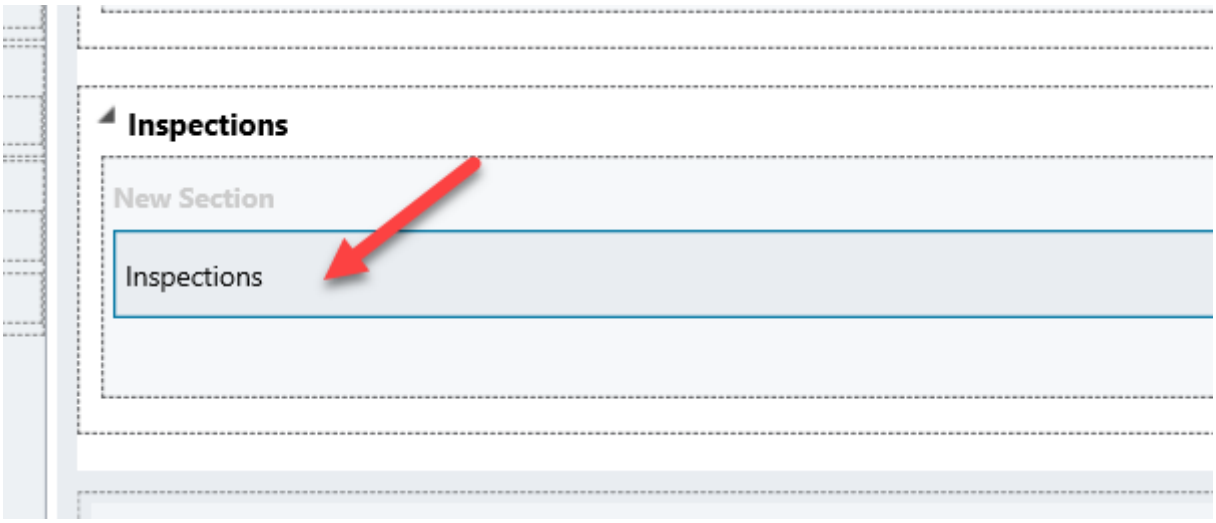
   - Select the **Forms** tab.
   - Click to open the **Main** form.



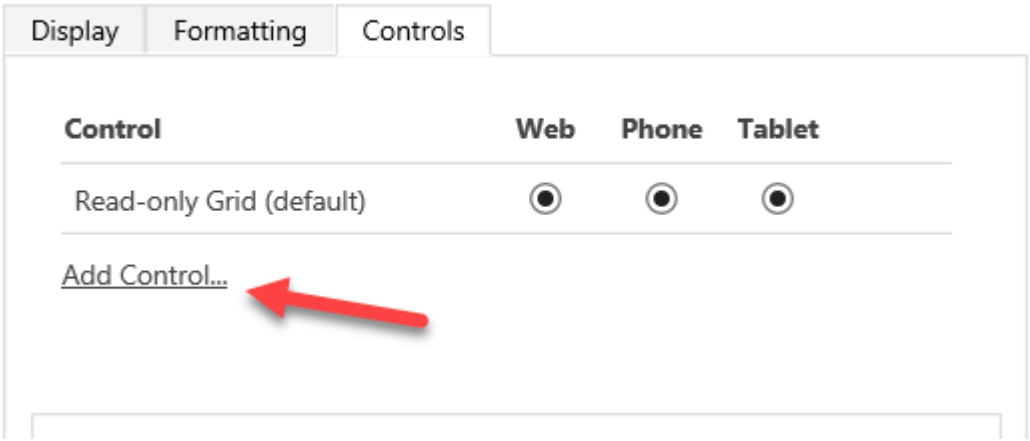
   - Click **Switch to Classic**.



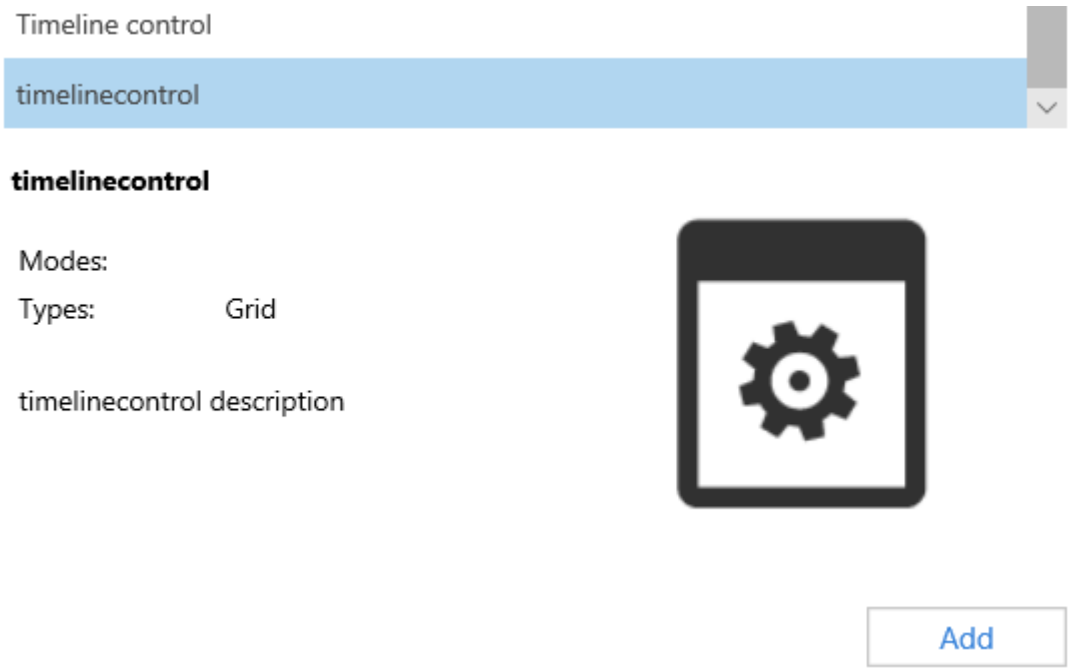
3. Add Timeline control to the form
   - Locate the **Inspections** tab.
   - Double click on the **Inspections** sub-grid.



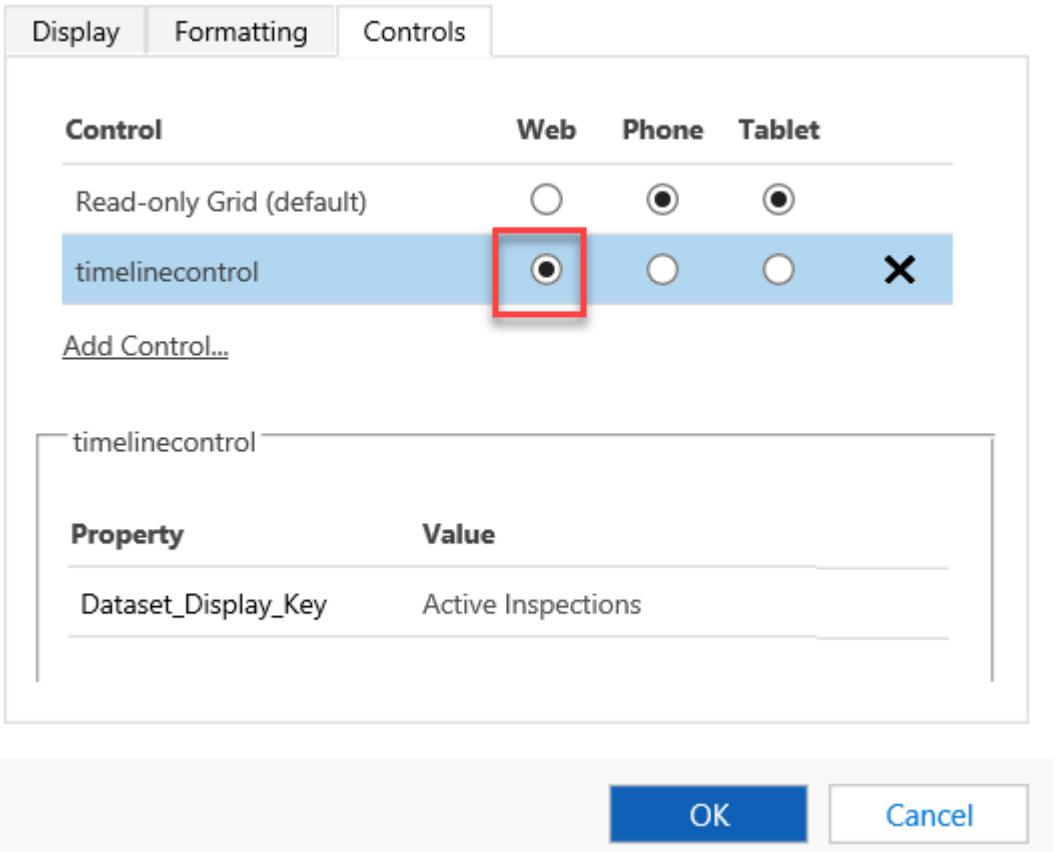
   - Select the **Controls** tab and click **Add Control**.



   - Select **timelinecontrol** and click Add.

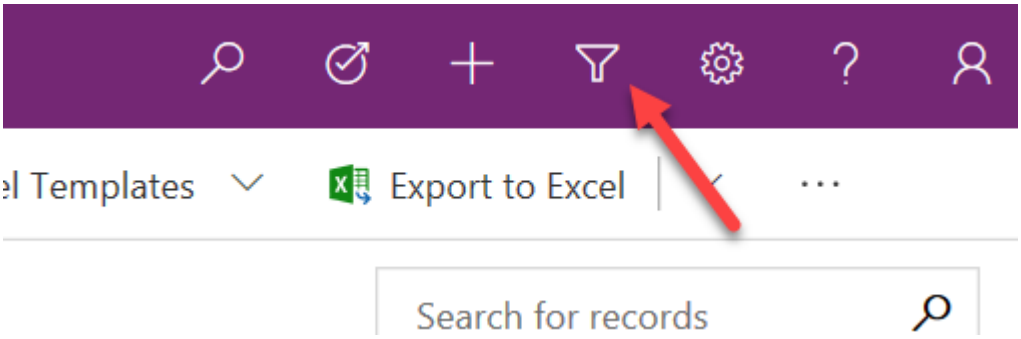


- Select **Web** and click **OK**.

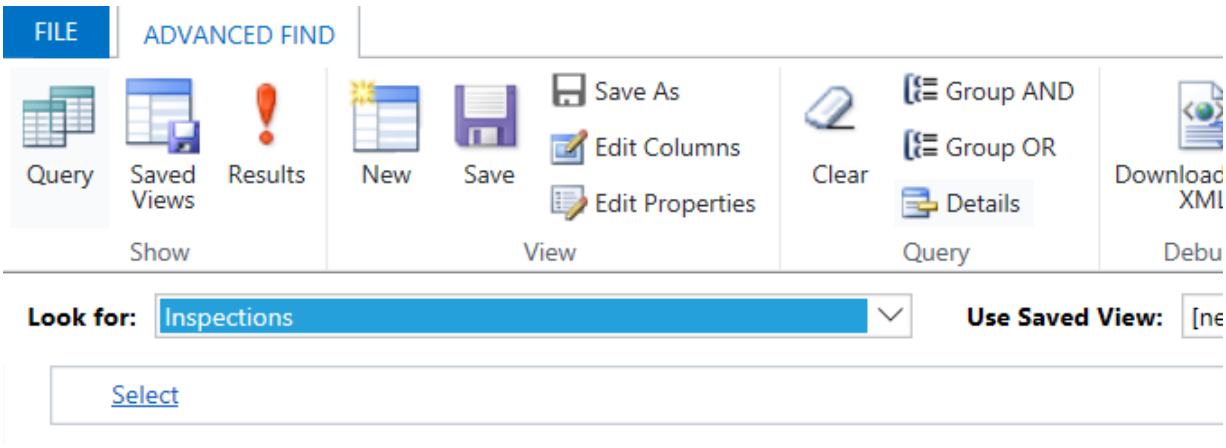


4. Save and publish
   - Click **Save**.
   - Click **Publish** and wait for the publishing to complete.
   - **DO NOT** close the form editor.
5. Create test data
   - Go back to Power Apps maker portal and make sure you have the **Dev** environment selected.
   - Select **Apps** and click to start the **Permit Management** application

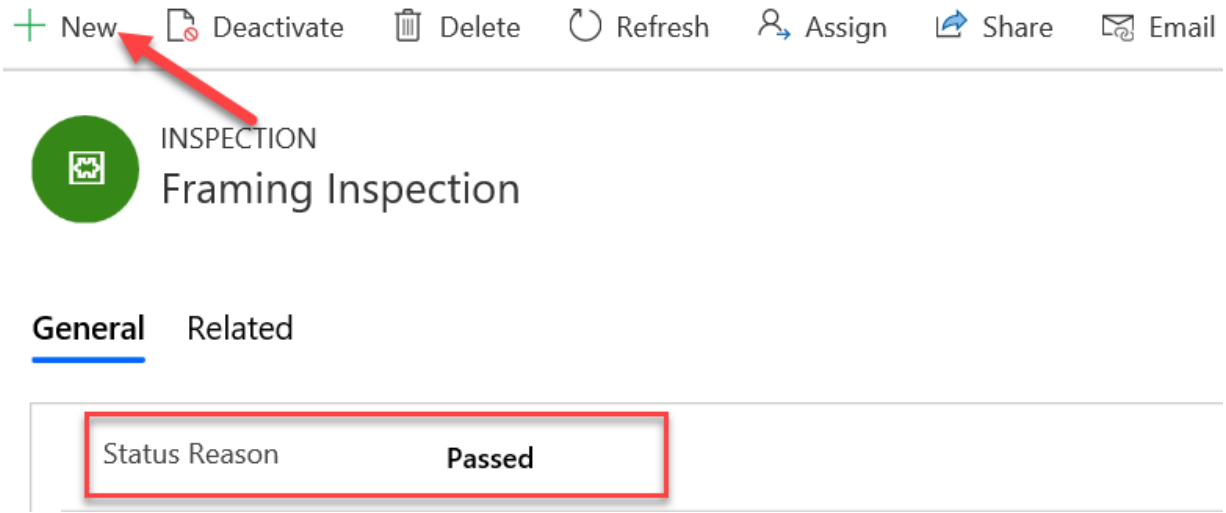
- Click **Advanced Find**.



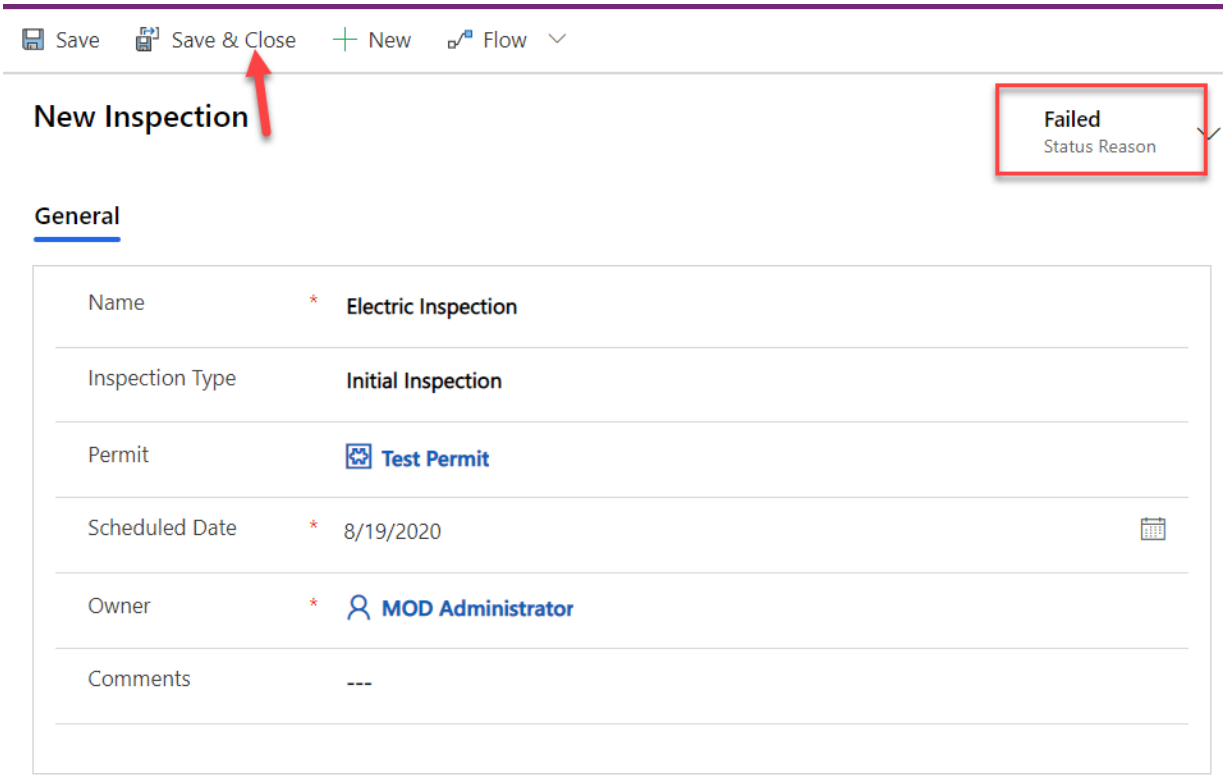
- Select **Inspections** and click **Results**.



- Click to open the **Framing Inspection**.
- Change the **Status Reason** to **Passed** and click **New**.

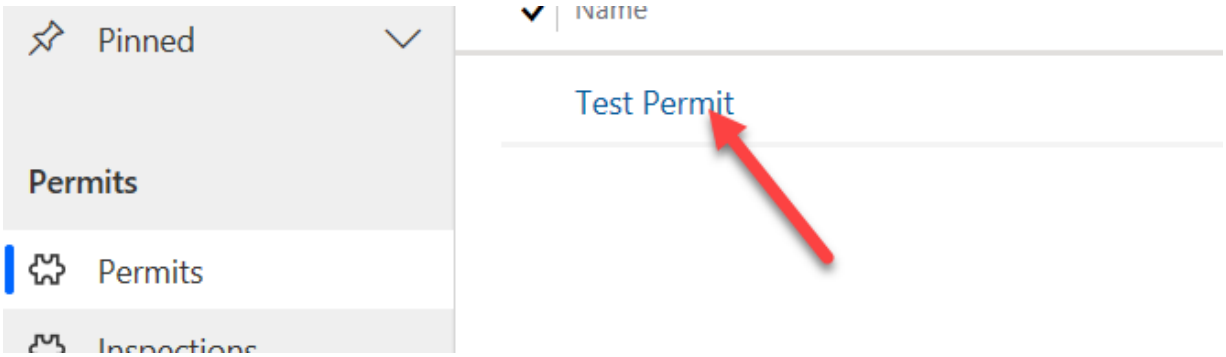


- Provide a Name, select Inspection Type, select the Test Permit, select Scheduled Date, select Failed for Status Reason, and click **Save and Close**.



- Close **Advanced Find**.

6. Test the control
   - Click to open a **Permit** record.



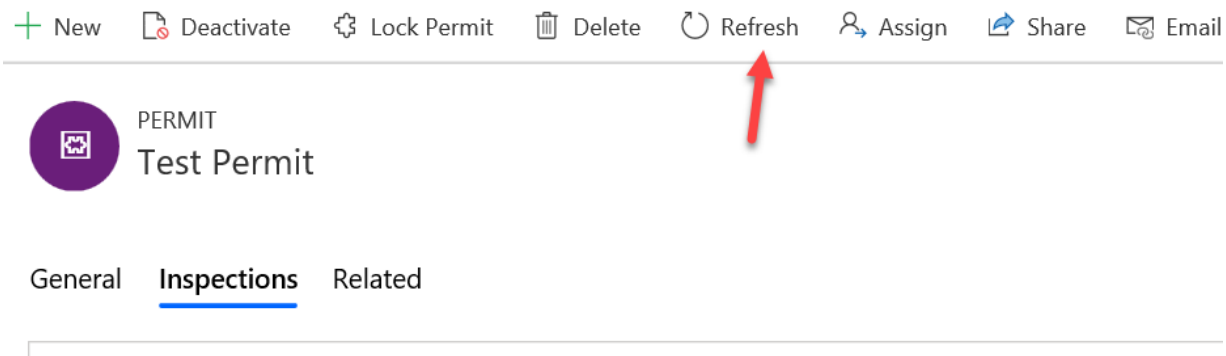
   - Select the **Inspections** tab.
   - The control should show the two inspections, but the color will not match the status reason values.

25.3 Task #3: Debug

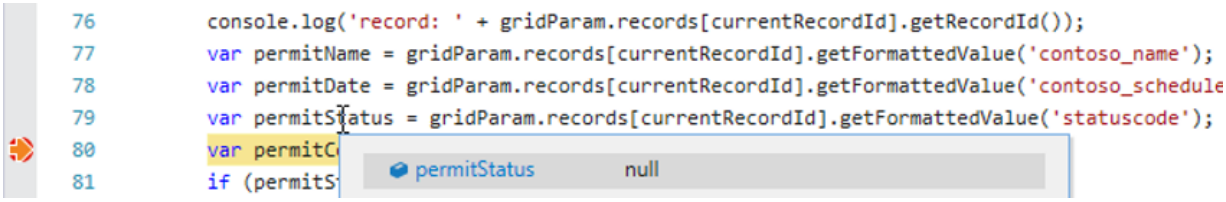
1. Start Edge DevTools and add breakpoint.
   - Press **F12**.
   - Search for **createTimelineData = function**
   - Locate the **createTimelineData** function and add breakpoint on the **permitColor =”green”** line.

```
69
70  timelinecontrol.prototype.createTimelineData = function (gridParam) {
71    this._timelineData = [];
72
73    if (gridParam.sortedRecordIds.length > 0) {
74      for (var _i = 0, _a = gridParam.sortedRecordIds; _i < _a.length; _i++) {
75        var currentRecordId = _a[_i];
76        console.log('record: ' + gridParam.records[currentRecordId].getRecordId());
77        var permitName = gridParam.records[currentRecordId].getFormattedValue('contoso_name');
78        var permitDate = gridParam.records[currentRecordId].getFormattedValue('contoso_scheduleddate');
79        var permitStatus = gridParam.records[currentRecordId].getFormattedValue('statuscode');
80        var permitColor = "green";
81        if (permitStatus == "Failed") permitColor = "red";
```

   - Go back to the Permit Management application and click Refresh.

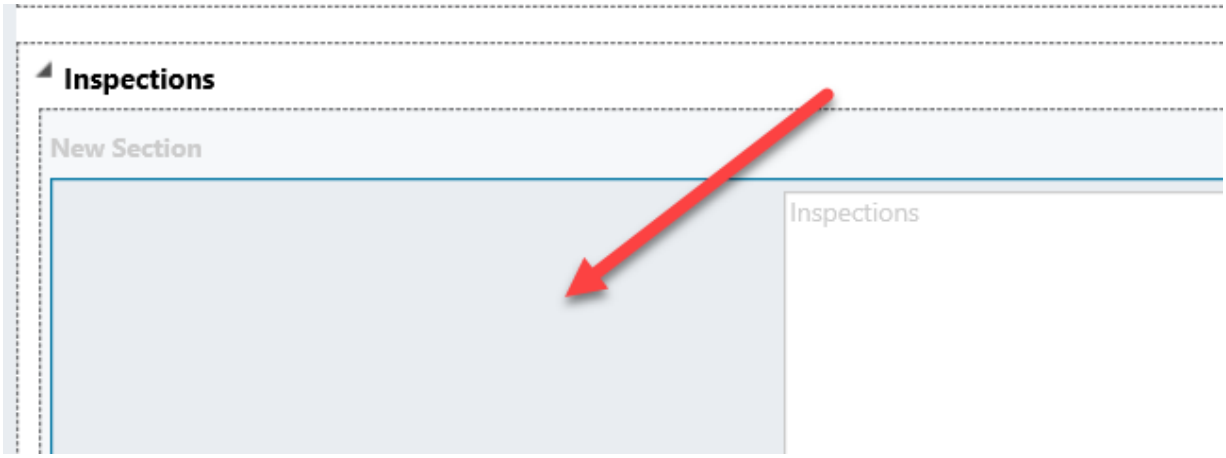


   - Select the **Inspections** tab again.
   - Execution should break.
   - Hover over the **permitStatus**, the **permitStatus** is null because **Status Reason** is not included in the **View**.

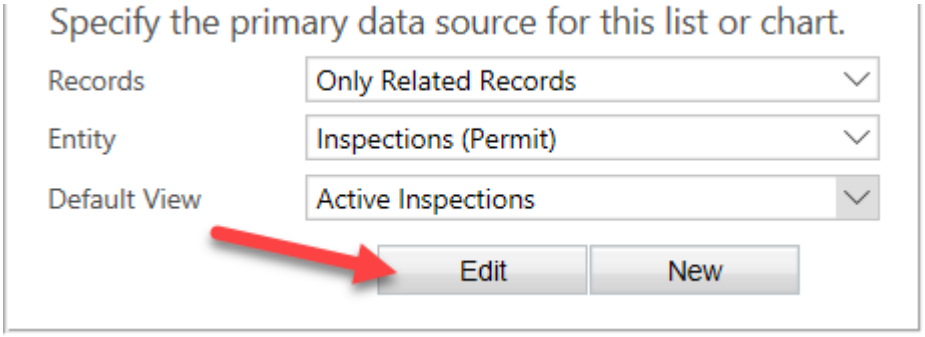


- Press **F5** to continue.
- Close the **DevTools**.

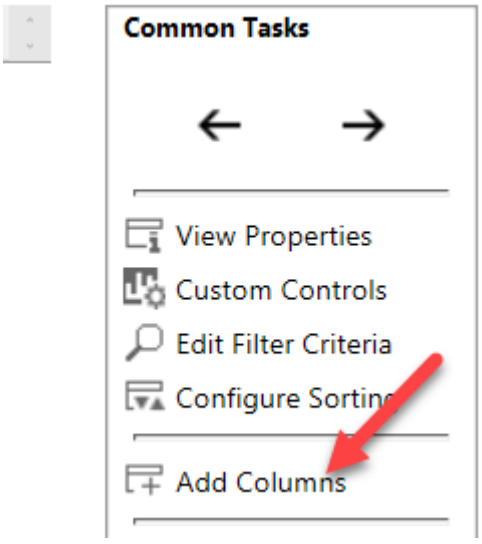
2. Add Status Reason to the view.
   - Go back to the form editor.
   - Double click on the Inspections timeline control.



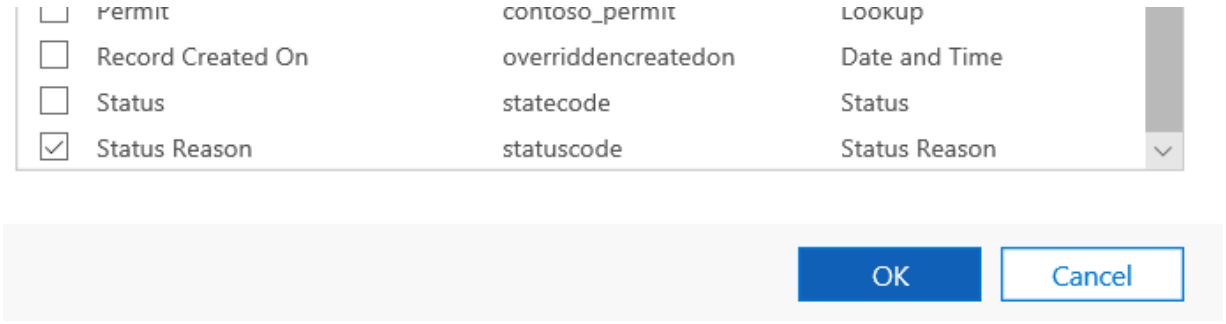
   - Click **Edit**.



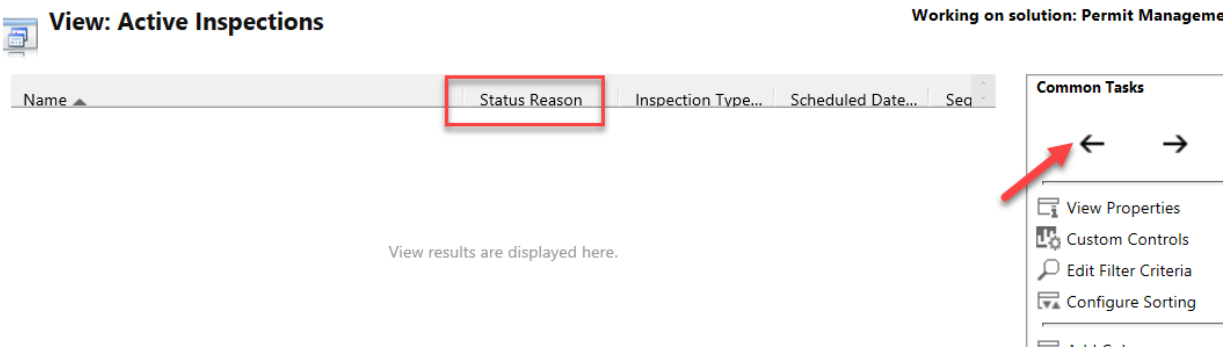
   - Click **Add Column**.



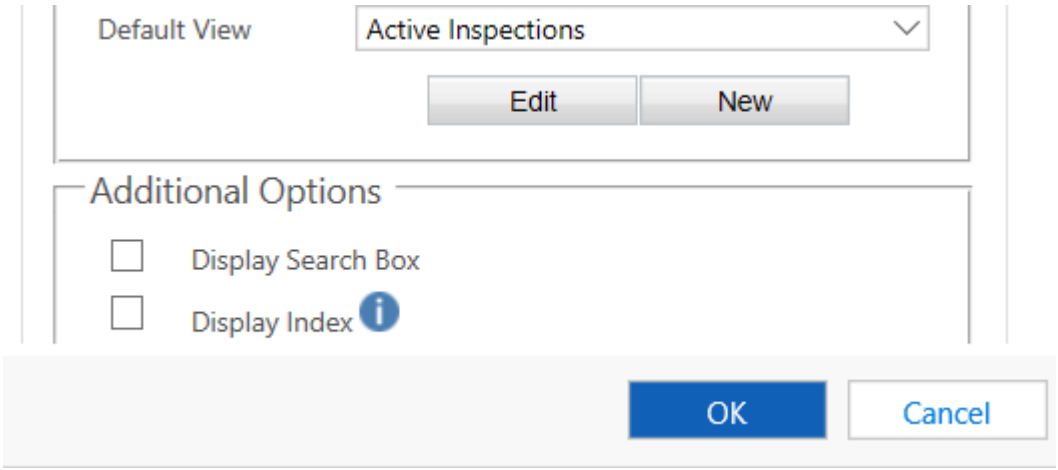
- Select **Status Reason** and click **OK**.



- Move the **Status Reason** Column after the **Name** Column.

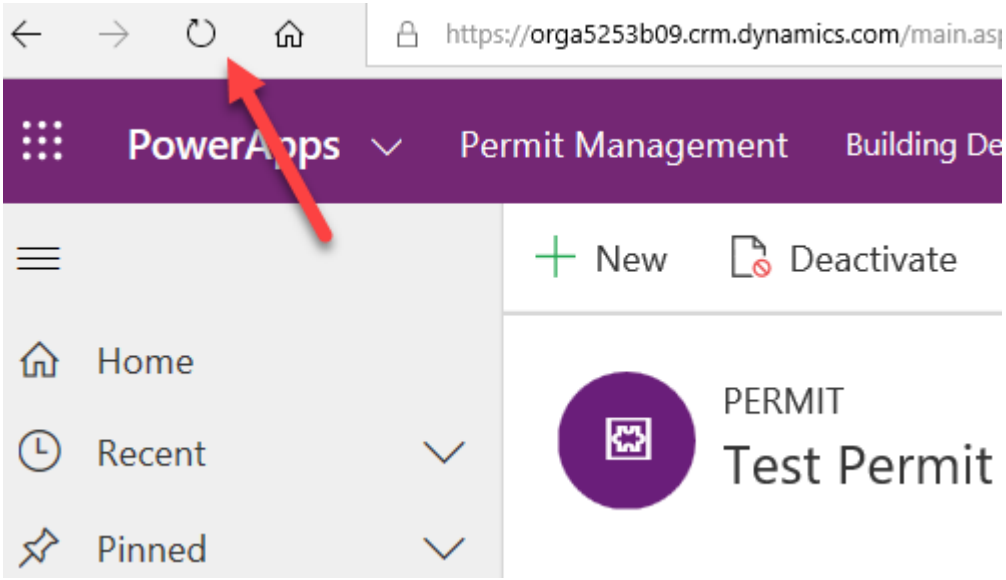


- Click **Save and Close**.
- Click **OK**.

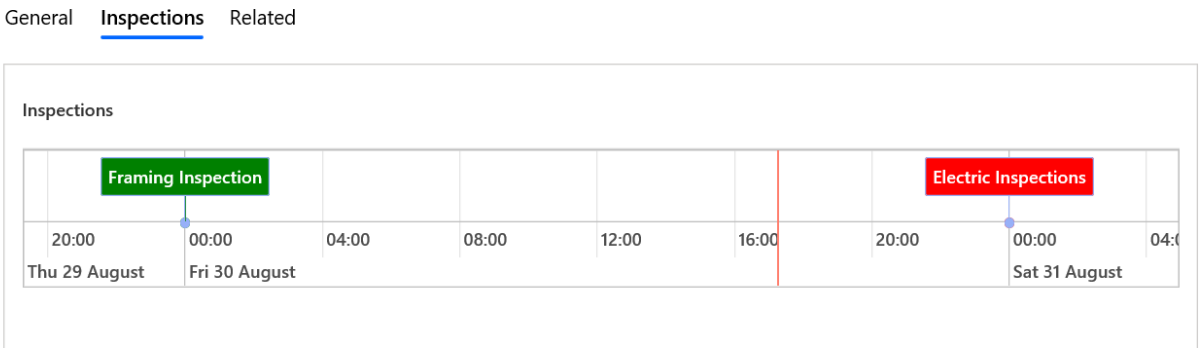


- Click **Save**.
- Click **Publish** and wait for the publishing to complete.
- Close the form editor.

3. Test your changes
   - Go back to the **Permit Management** application and refresh the browser.

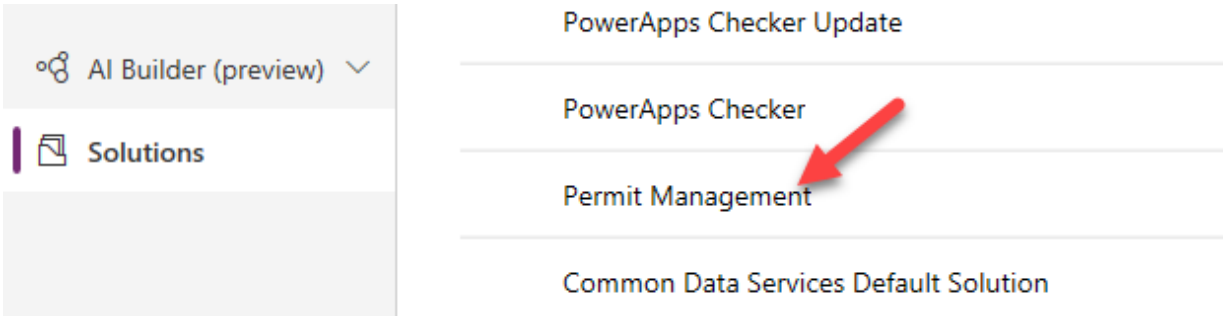


- Select the **Inspections** tab. The timeline control should now show the correct colors.

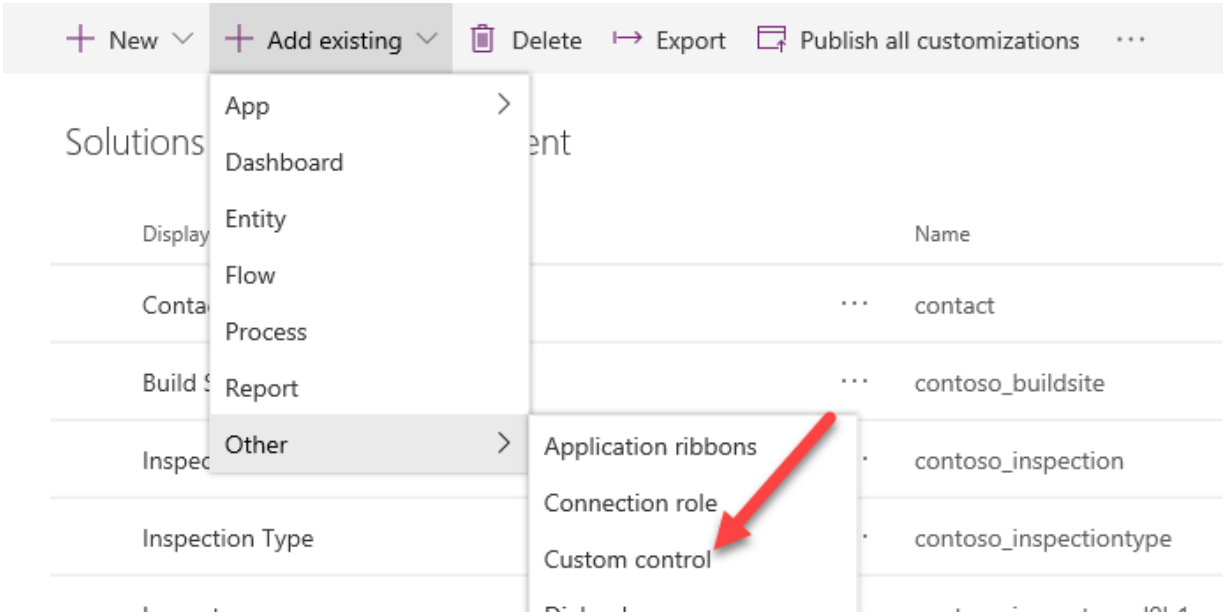


25.4 Task #4: Add the Timeline Control to Permit Management Solution

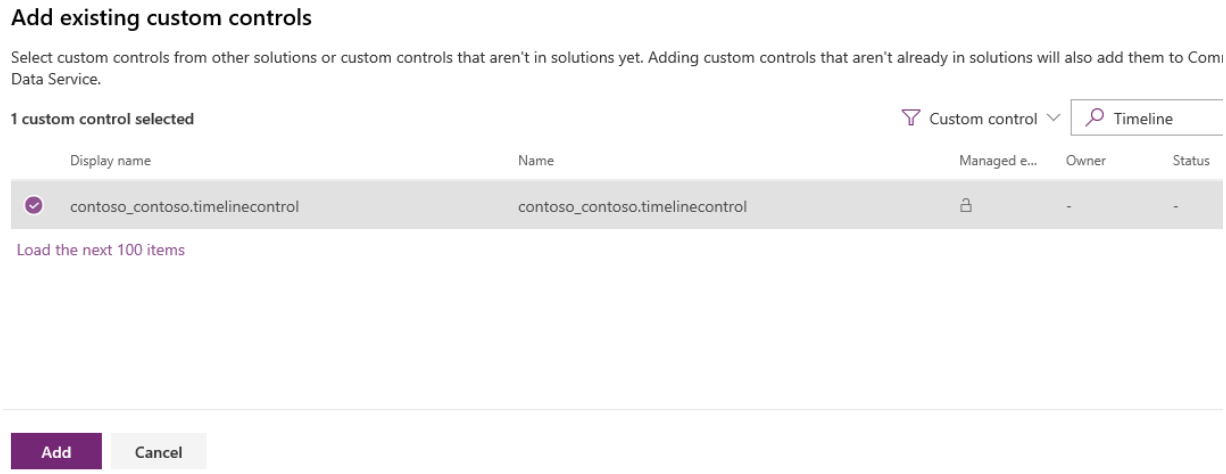
1. Add Custom Control to solution
   - Navigate to Power Apps maker portal - screenshot and make sure you are in the **Dev** environment.
   - Select **Solutions** and click to open the **Permit Management** solution.



   - Click **Add Existing | Other | Custom Control**.



- Search for Timeline, select **contoso_contoso.timelinecontrol** and click **Add**.



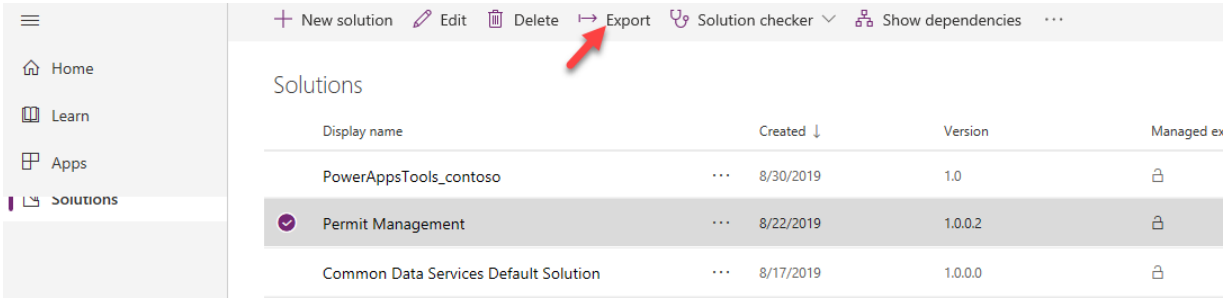
- Click **Publish All Customizations** and wait for the publishing to complete.

26 Exercise #3: Promote to production

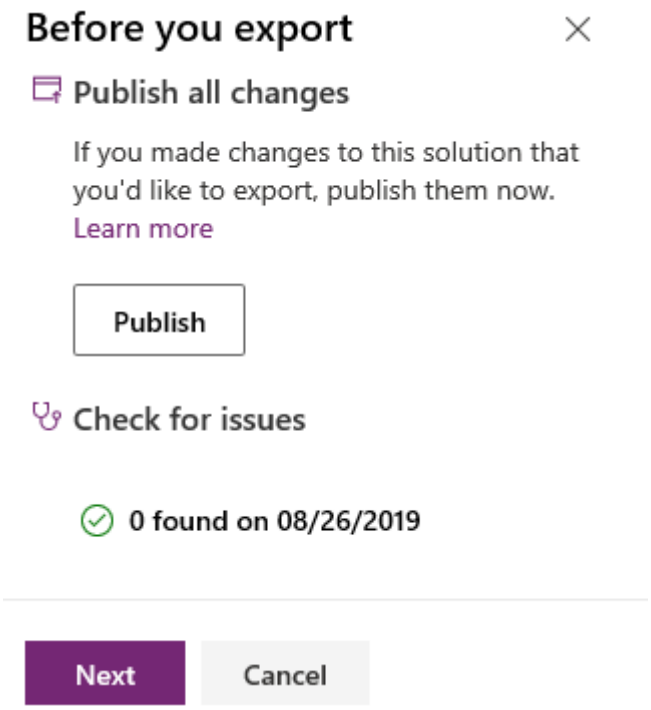
Objective: In this exercise, you will export the Permit Management solution form your Dev environment and import it into your Production environment.

26.1 Task #1: Export Solution

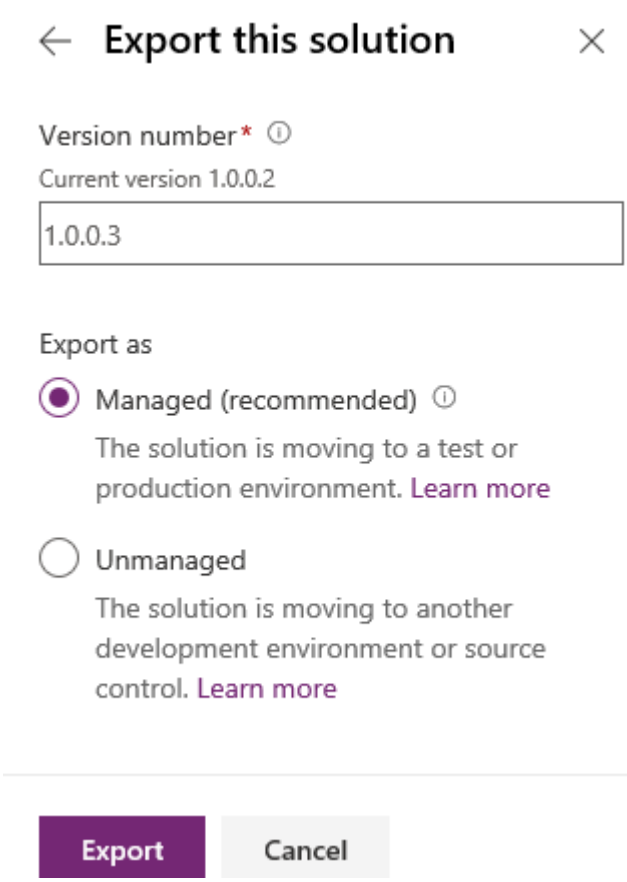
1. Export Permit Management managed solution
   - Log on to Power Apps maker portal and make sure you are in the **Dev** environment.
   - Select **Solution**.
   - Select the **Permit Management** solution and click **Export**.



- Click **Publish** and wait for the publishing to complete.



- Click **Next**.
- Select **Managed** and click **Export**.

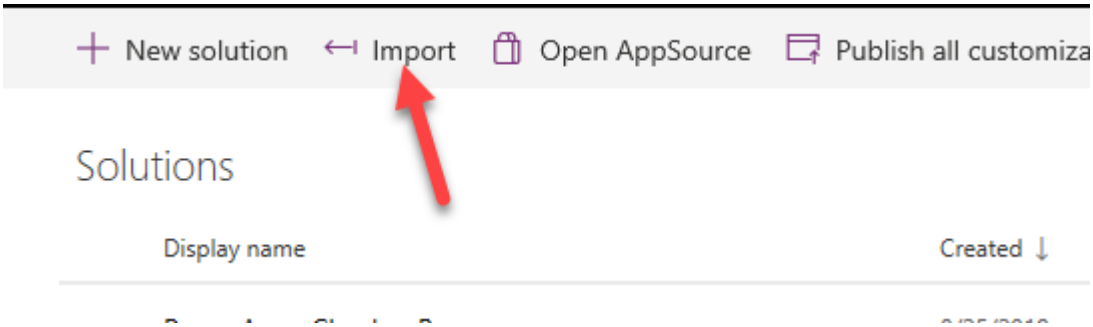


- Save the **Exported** solution on your machine.

2. Export Permit Management unmanaged solution
   - Select **Solution** again.
   - Select the **Permit Management** solution and click **Export**.
   - Click **Next**.
   - Select **Unmanaged, edit the version number** to match the Managed Solution you just exported and click **Export**.
   - Save the **Exported** solution on your machine.

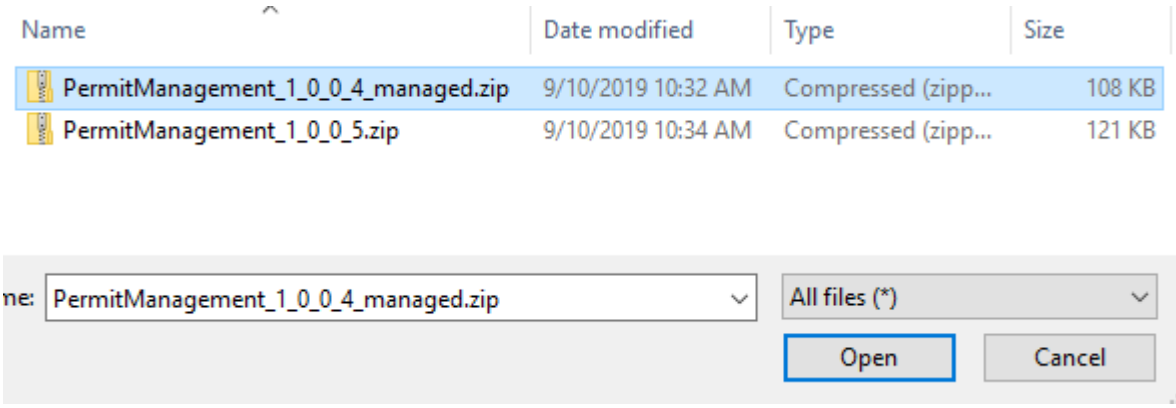
26.2 Task #2: Import Solution

1. Import Permit Management managed solution
   - Log on to Powe Apps maker portal and make sure you are in the **Prod** environment.
   - Select **Solution**.
   - Click **Import**.

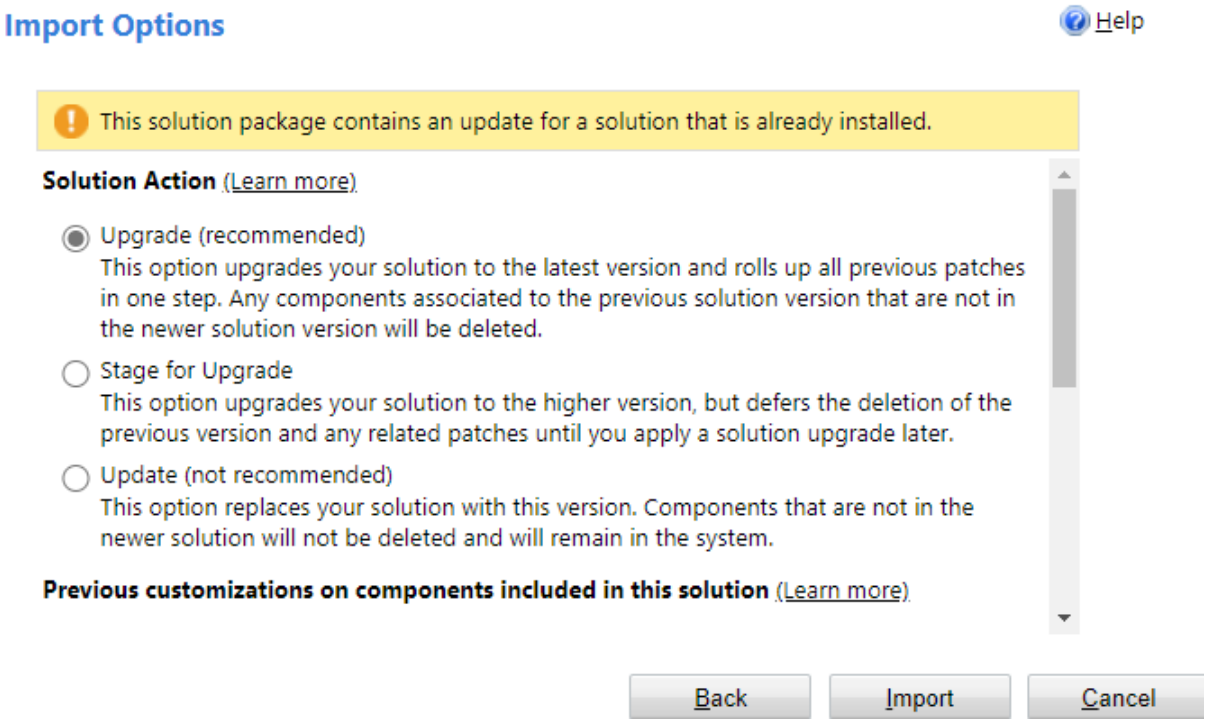


   - Click **Choose File**.

- Select the **Managed** solution you exported and click **Open**.



- Click **Next**.
- Click **Next** again.
- Select the **Upgrade** option if you already have another version of the same solution.
- Click **Import**.



Wait for the import to complete and click **Close**

26.3 Review the production application by adding a few records and testing your progress.

26.4 lab: title: 'Lab 06: Plug-ins'

[!NOTE] Effective November 2020:

- Common Data Service has been renamed to Microsoft Dataverse. Learn more
- Some terminology in Microsoft Dataverse has been updated. For example, *Table* is now *table* and *Column* is now *column*. Learn more

This content will be updated soon to reflect the latest terminology.

26.5 Lab 06 – Plug-ins

27 Scenario

A regional building department issues and tracks permits for new buildings and updates for remodeling of existing buildings. Throughout this course you will build applications and automation to enable the regional building department to manage the permitting process. This will be an end-to-end solution which will help you understand the overall process flow.

In this lab you will build two plugins. The first plugin will run when a new permit record is created, and it will check that there are no other permits that exists for the build sites that are "locked". If a locked permit is found, the plugin will block the creation of this new permit. The second plugin will hook up and run when the Lock Permit custom action is invoked. In the prior module we defined the custom action, and now with the plugin step registered on execution of the custom action, it will now perform the lock permit business logic.

28 High-level lab steps

As part of building the plugins, you will complete the following activities.

- Create two plugins using a common provided base class
- Implement logic to work with a plugin registered on an Table event
- Implement logic to work with a plugin registered on a custom action event
- Deploy the plugins and associate them with your solution
- Use the plugin trace log to see traces from the plugin
- Debug the plugin on your local computer

28.1 Things to consider before you begin

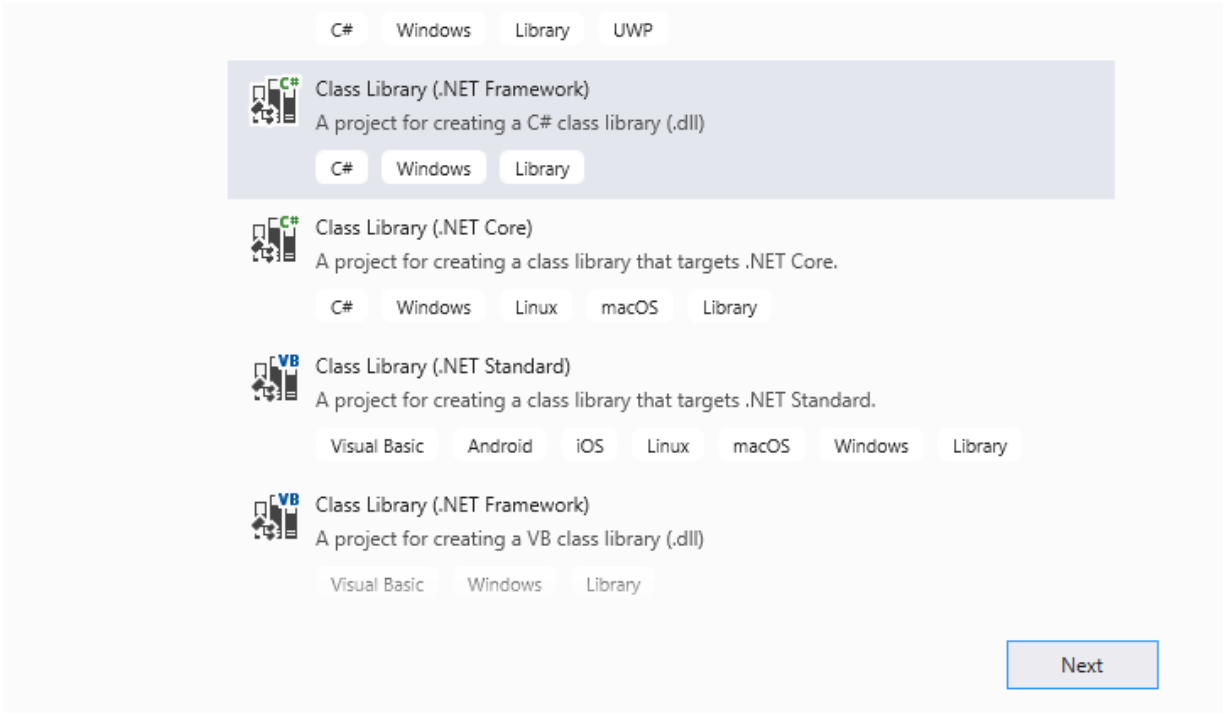
- Do we know what events will trigger our plugins?
- Could what we are doing with the plugin, be done using Power Automate?
- Remember to continue working in your DEVELOPMENT environment. We'll move everything to production soon.

29 Exercise #1: Block New Permit Creation Plugin

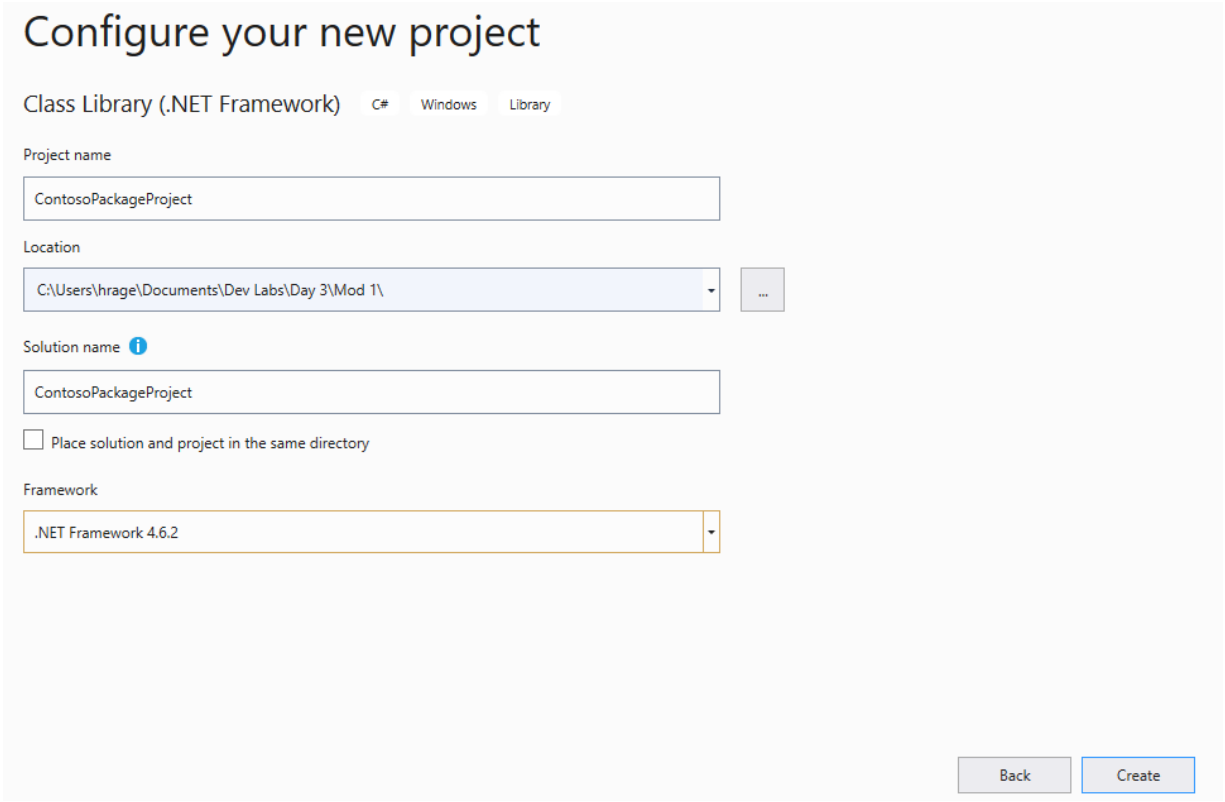
Objective: In this exercise, you will create a plugin that will run on create permit. This plugin will check if there are any locked permits for the selected build site of the new permit and block the creation of the new permit.

29.1 Task #1: Create the plugin

1. Create a Visual Studio project
    - Start **Visual Studio**.
    - Click **File > New > Project**.
    - Select **Class Library (.NET Framework)** and click **Next**. Make sure you have selected the one with C#.

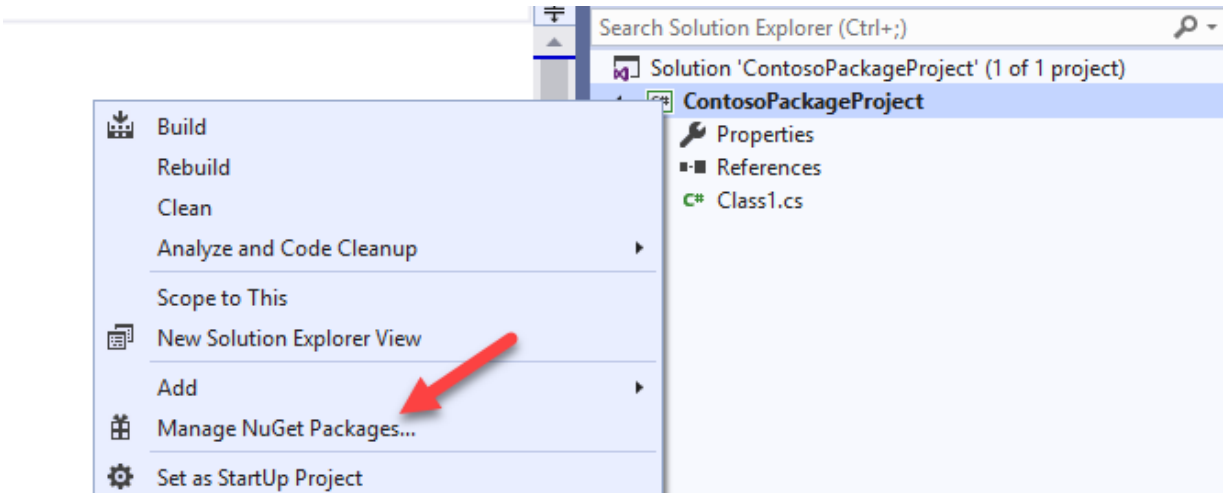


- Enter **ContosoPackageProject** for **Project Name**, select a location to save the project, select **.NET Framework 4.6.2** for **Framework**, and then select **Create**.

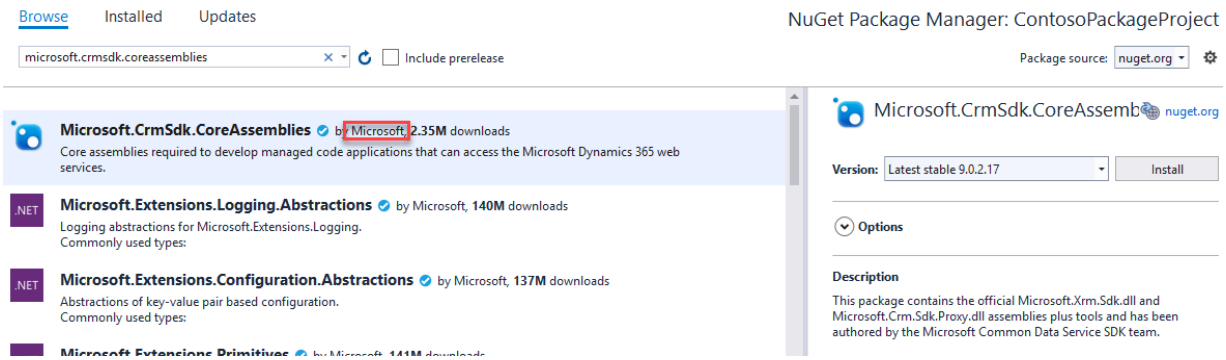


2. Add NuGet packages

   - Right click on the project name and select **Manage NuGet Packages**.

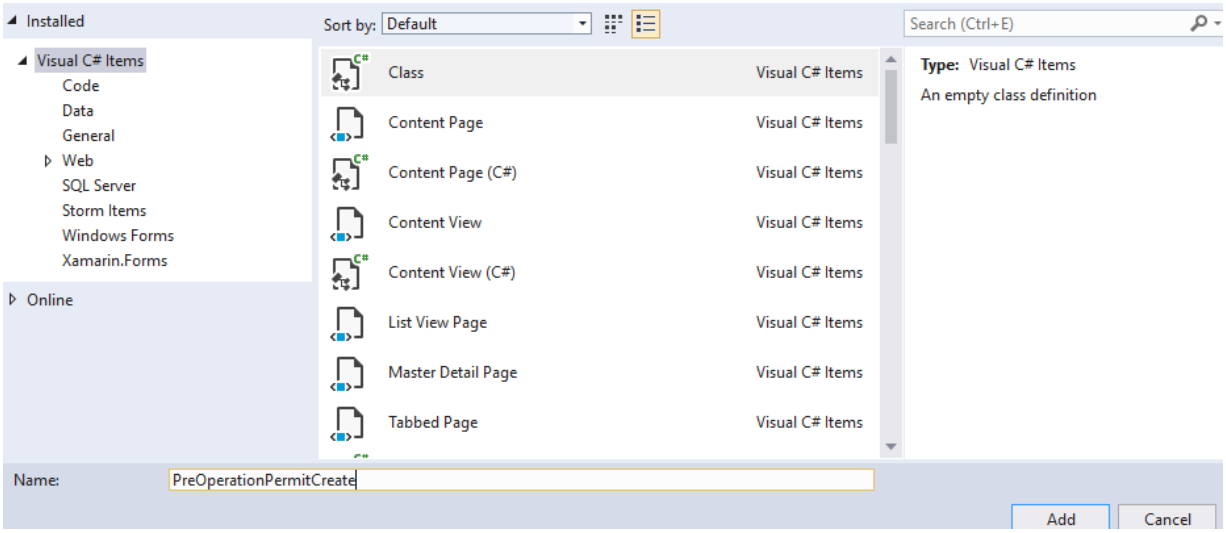


- Select the **Browse** tab.
- Search for **microsoft.CrmSdk.CoreAssemblies**.
- Select **Microsoft.crmsdk.coreassemblies** (one authored by Microsoft)and click **Install**.



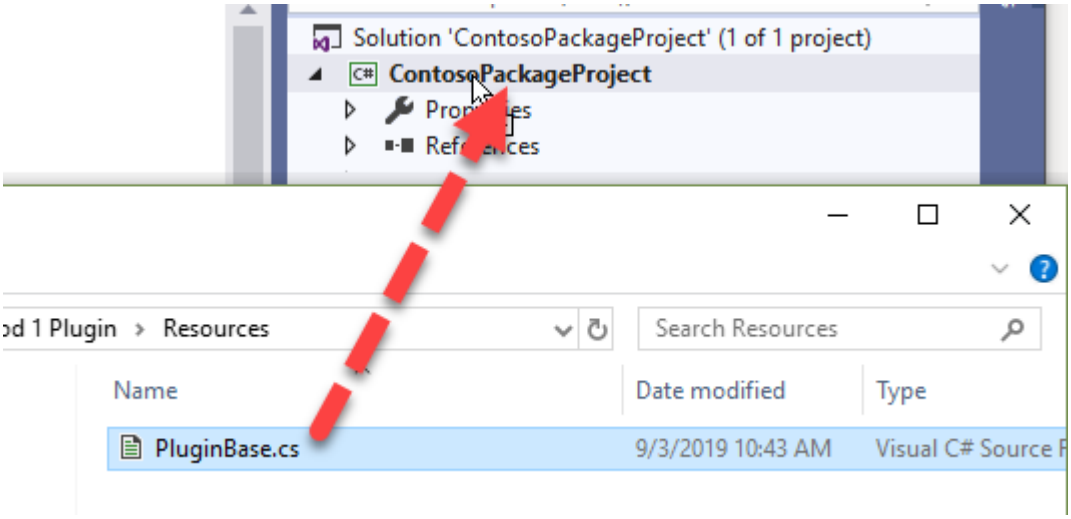
- Read the license terms and then select **I agree** if you agree.

3. Delete Class1 and create new class file for the plugin
   - Right click on **Class1.cs** and **Delete.** Click **OK** on the warning to delete this permanently.
   - Right on the project and select **Add > Class**.
   - Enter **PreOperationPermitCreate** for **Name** and click **Add**.



4. Add **PluginBase** file to the project.
   - Locate the **PluginBase** file in the lab resources folder.

- Drag the **PluginBase** file to your Visual Studio project. Make sure that the file now exists in the folder structure.



- Review the **PluginBase** class. This class inherits from IPlugin.

```
using System.ServiceModel;
using Microsoft.Xrm.Sdk;

/// <summary>
/// Base class for all Plug-in classes.
/// </summary>
0 references
public abstract class PluginBase : IPlugin
{
    /// <summary>
    /// Plug-in Context object.
    /// </summary>
    5 references
```

Close the **PluginBase** file.

5. Add **using statements** to the **PreOperationPermitCreate** class, make the class **public**, and inherit from **PuginBase**

- Add the using statement below to the **PreOperationPermitCreate** class.

```
using Microsoft.Xrm.Sdk;
using Microsoft.Xrm.Sdk.Query;
using ContosoPackagePoject;
```

- Make the **PreOperationPermitCreate** public and **inherit** from **PluginBase**.

```
using System.Threading.Tasks;

using Microsoft.Xrm.Sdk;
using Microsoft.Xrm.Sdk.Query;

namespace ContosoPackageProject
{
    0 references
    public class PreOperationPermitCreate : PluginBase
    {

    }
}
```

6. Override the ExecuteCDSPlugin method and get the Target Table and the Build Site Table reference.

- To override the **ExecuteCDSPlugin** method, add the code below to the **PreOperationPermitCreate** method.

```
protected override void ExecuteCDSPlugin(LocalPluginContext localcontext)

{

base.ExecuteCDSPlugin(localcontext);

}
```

- To get the **Target** Table, add the code below inside the ExecuteCDSPlugin method.

```
var permitEntity = localcontext.PluginExecutionContext.InputParameters["Target"] as Entity;
```

- To get the Build Site Table reference, add the below code after **permitEntity** variable definition.

```
var buildSiteRef = permitEntity["contoso_buildsite"] as EntityReference;
```

- To add Trace Messages, add the below mentioned code after **buildSiteRef** variable definition.

```
localcontext.Trace("Primary Entity Id: " + permitEntity.Id);

localcontext.Trace("Build Site Entity Id: " + buildSiteRef.Id);
```

7. Create Fetch xml and that will get the count of locked permits matching the build site id and call retrieve multiple.
   - Create the **FetchXML** string.

     ```
     string fetchString = "<fetch output-format='xml-platform' distinct='false' version='1.0'
     ```

   - Call RetrieveMultiple and add Trace Message.

     ```
     localcontext.Trace("Calling RetrieveMultiple for locked permits");
     var response = localcontext.OrganizationService.RetrieveMultiple(new FetchExpression(fetc
     ```

8. Get the locked Permit Count and throw InvalidPluginExecutionException if the **Count** is more than 0
   - Get the locker permits **Count**.

     ```
     int lockedPermitCount = (int)((AliasedValue)response.Tables[0]["Count"]).Value;
     ```

- Add Trace Message, check if the **Count** is more than **0** and throw **InvalidPluginExecutionException** if it is more than **0**.

```
localcontext.Trace("Locket Permit count : " + lockedPermitCount);
if (lockedPermitCount > 0)
{
throw new InvalidPluginExecutionException("Too many locked permits for build site");
}
```

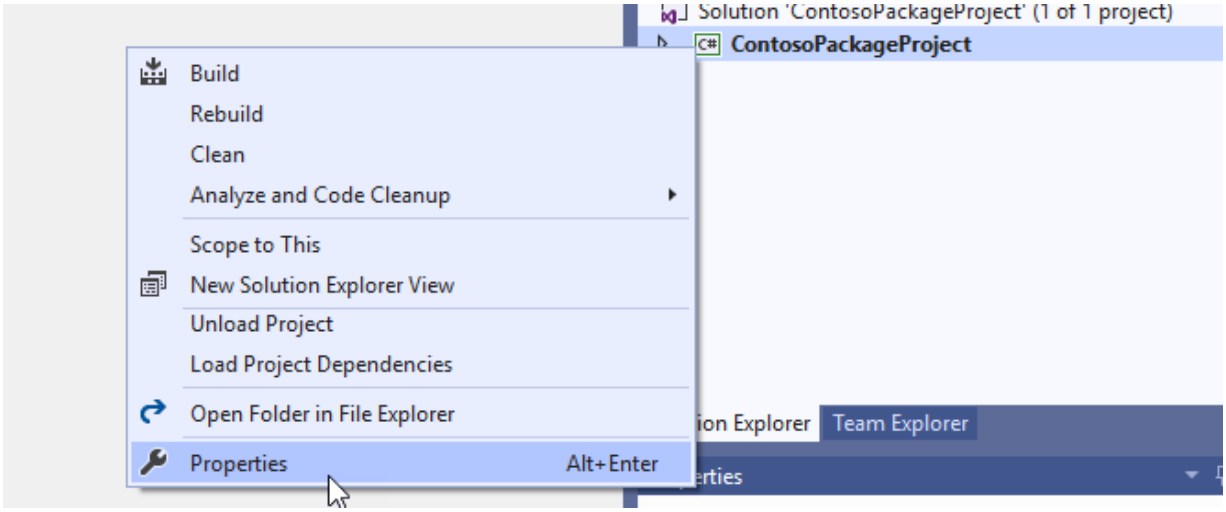
- The ExecuteCDSPlugin method should now look like the image below.

```
3 references
protected override void ExecuteCDSPlugin(LocalPluginContext localcontext)
{
    base.ExecuteCDSPlugin(localcontext);

    var permitEntity = localcontext.PluginExecutionContext.InputParameters["Target"] as Entity;
    var buildSiteRef = permitEntity["contoso_buildsite"] as EntityReference;
    localcontext.Trace("Primary Entity Id: " + permitEntity.Id);
    localcontext.Trace("Build Site Entity Id: " + buildSiteRef.Id);
    string fetchString = "<fetch output-format='xml-platform' distinct='false' version='1.0'  mapping='
    localcontext.Trace("Calling RetrieveMultiple for locked permits");
    var response = localcontext.OrganizationService.RetrieveMultiple(new FetchExpression(fetchString));
    int lockedPermitCount = (int)((AliasedValue)response.Entities[0]["Count"]).Value;
    localcontext.Trace("Locket Permit count : " + lockedPermitCount);
    if (lockedPermitCount > 0)
    {
        throw new InvalidPluginExecutionException("Too many locked permits for build site");
    }
}
```

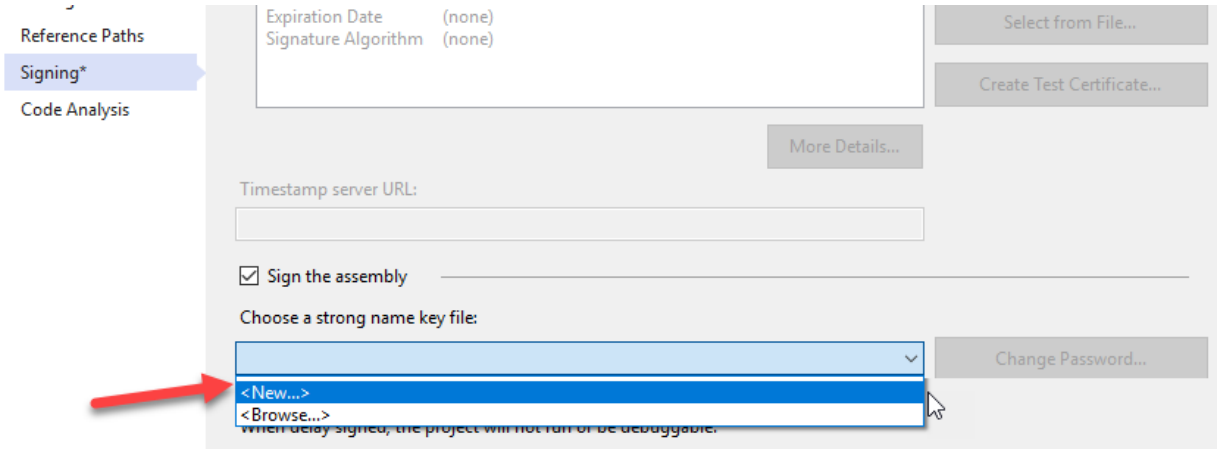
- Build the project and make sure it succeeds. To build the project, right click on the project and select **Build**. Check the output and make sure that the build is succeeded. If it does not, go back and review your work compared the steps documented here.

29.2 Task #2: Deploy the plugin

1. Create strong name key.
   - Right click on the **Project** and select **Properties**.

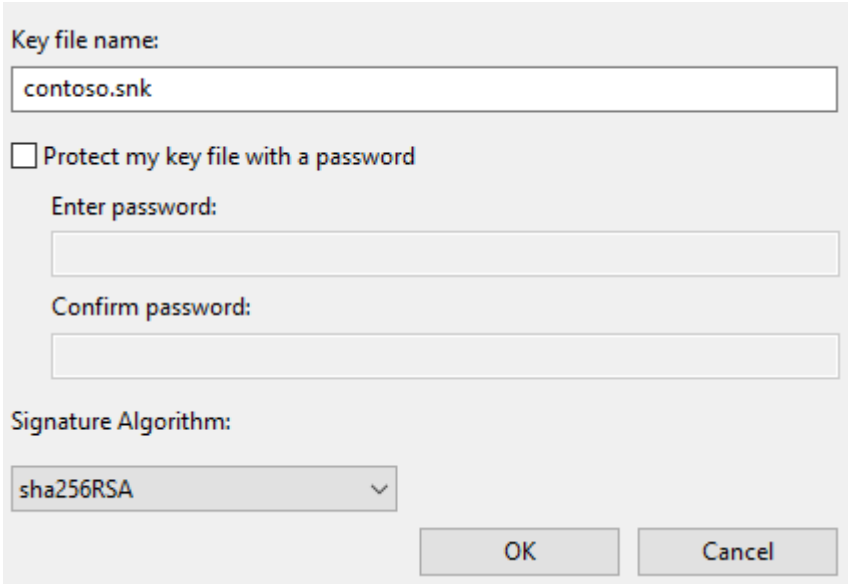


   - Select the **Signing** tab, check the **Sign the assembly** checkbox and select **<New...>**.



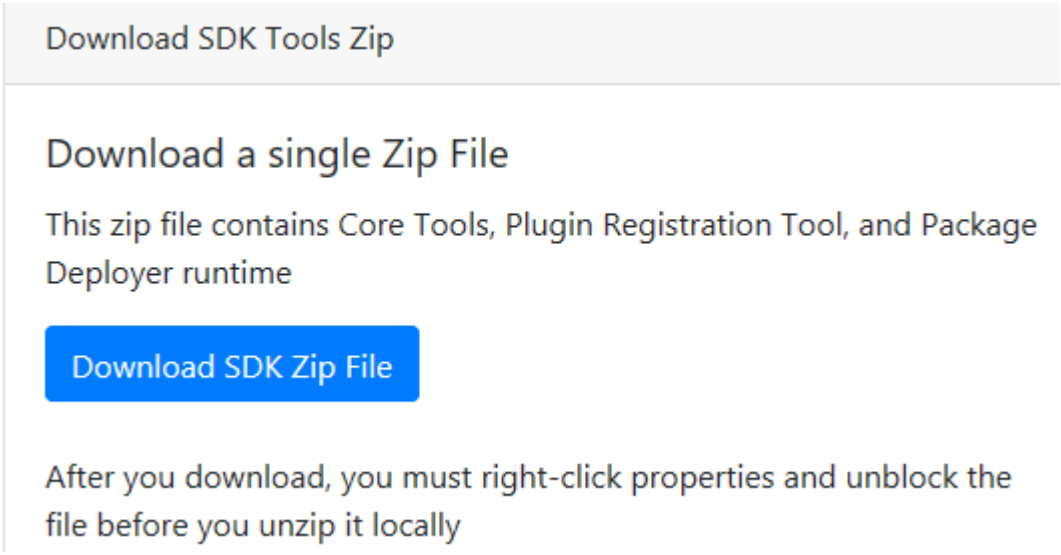
   - Enter **contoso.snk** for **Name**, uncheck the Protect with a **Password** checkbox, and click **OK**. Note: In case you get an access denied while creating the signature, close Visual Studio and run it

in administrator mode to successfully complete this step.

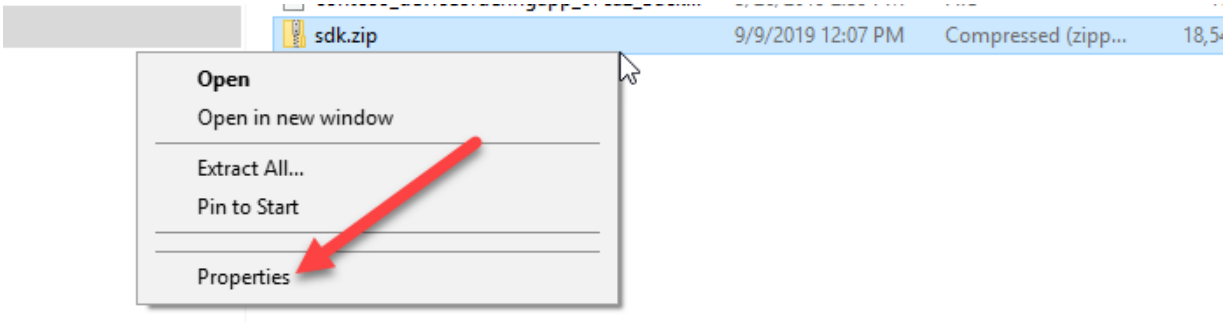


- Rebuild the project. Make sure the project is rebuilt successfully.

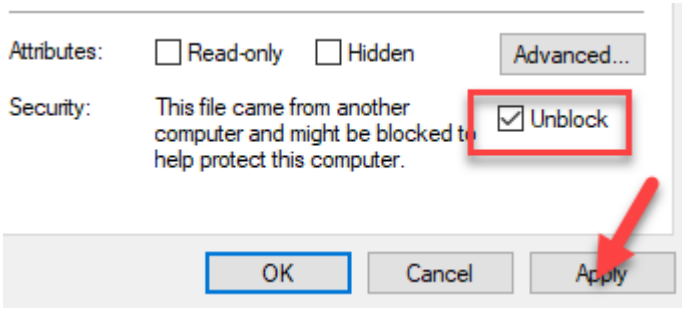
2. If you don't have CDS/Dynamics 365 SDK tools downloaded already, download them using the following method:
   - Navigate to https://xrm.tools/SDK
   - Click **Download SDK Zip File**.



   - Save the zip file on your machine.
   - Right click on the downloaded **sdk.zip** file and select **Properties**.

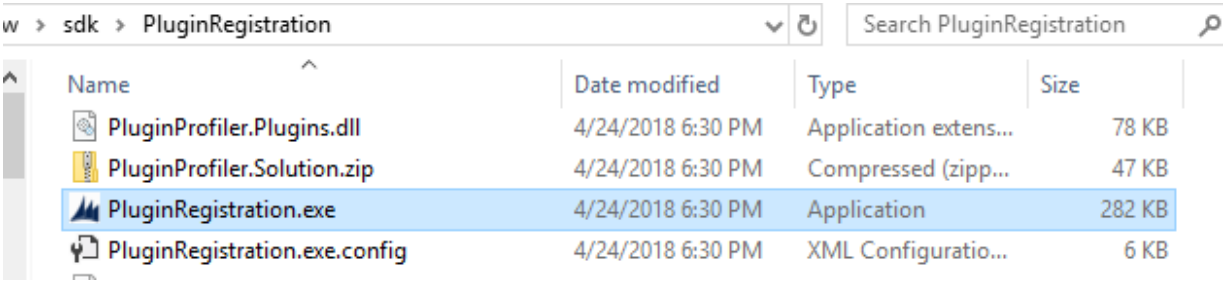


   - Check the **Unblock** checkbox and click Apply.

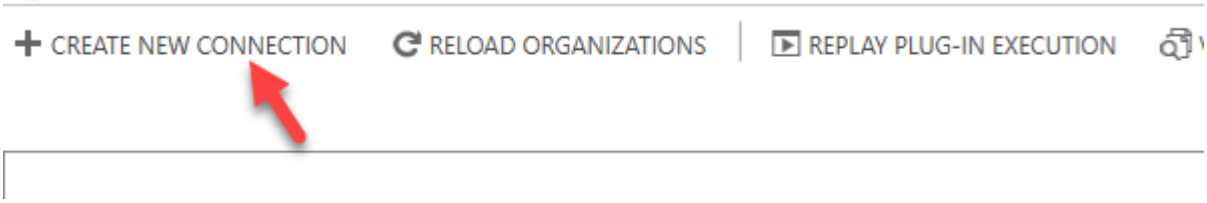


- Click **OK**.
- Right click on the **sdk.zip** file again and select **Extract All**.
- Complete extracting.

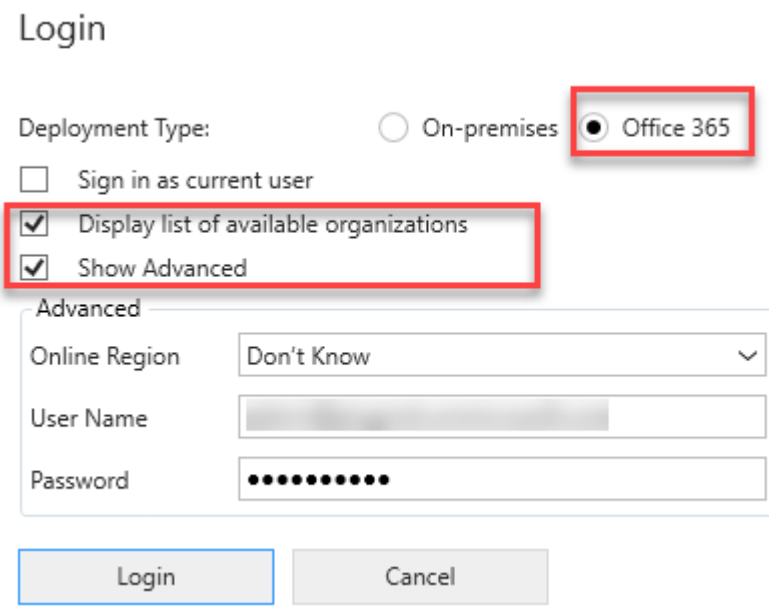
3. Start the plugin registration tool and sign in.
   - Open the **sdk** folder you extracted and click to open the **PluginRegistration** folder.
   - Locate **PluginRegistration.exe** and double click to start. This will open a new window.



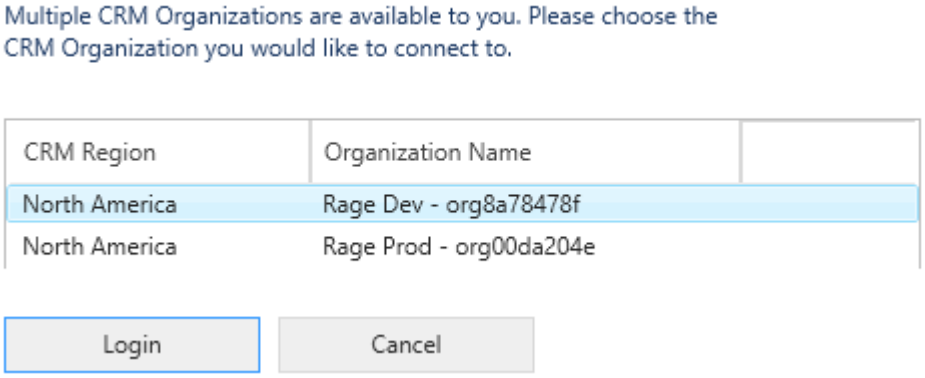
4. Connect to your org.
   - Click **Create New Connection**.



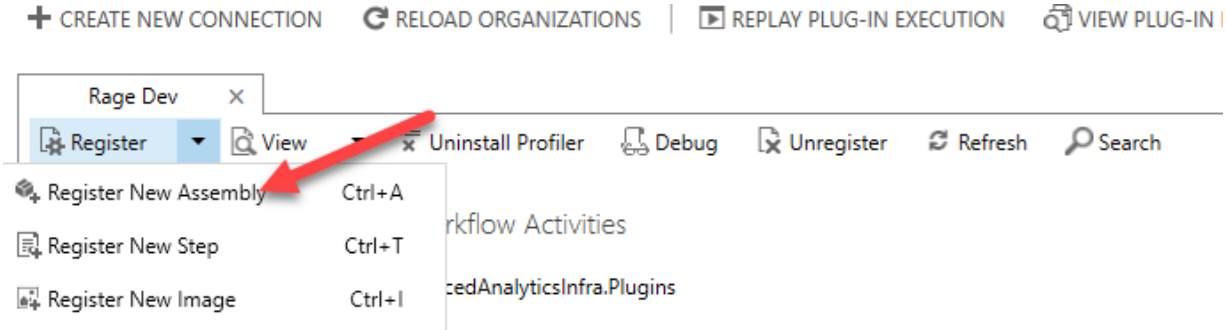
- Select **Office 365** and check the **Display List of available organization** and **Show Advanced** checkboxes. Select **Online Region** where your organization is located. If you are unsure what region to select, select **Don't Know**.
- Provide your **Microsoft Dataverse** credentials and click **Login**.



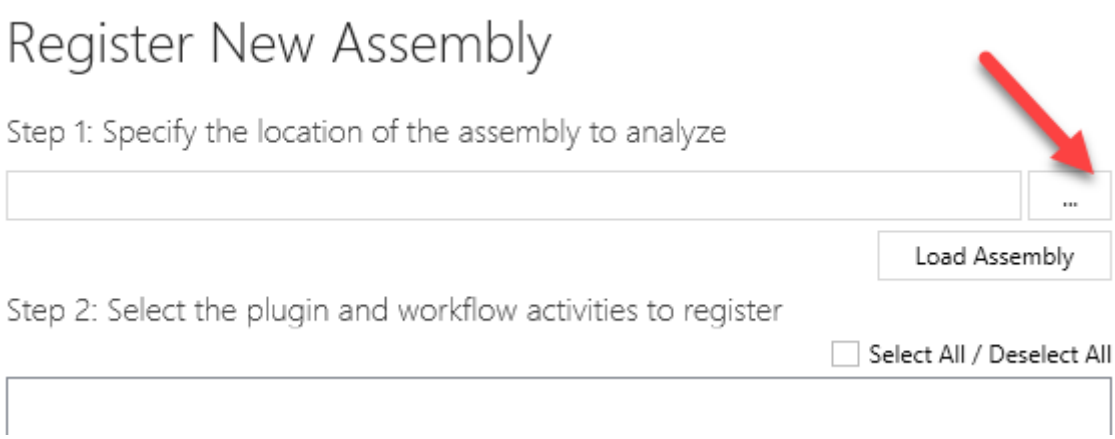
– Select the **Dev** environment and click **Login**.



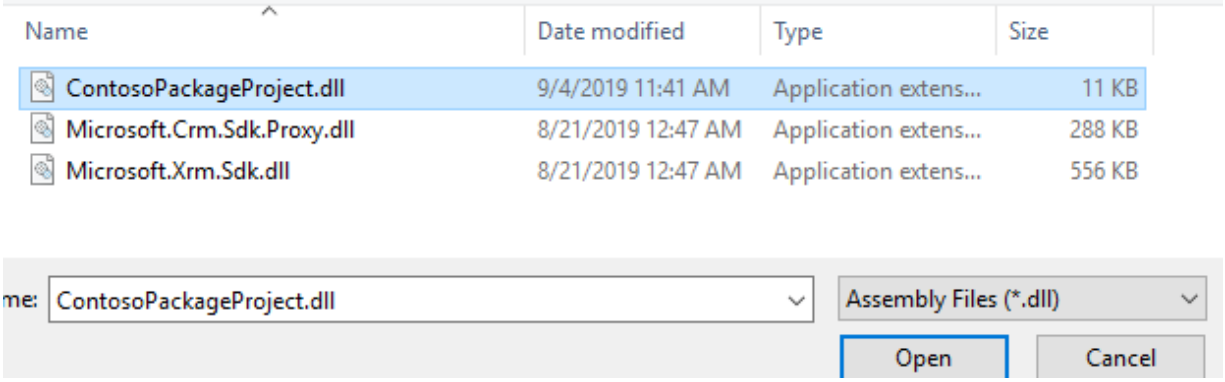
4. Register new assembly
   - Click **Register** and select **Register New Assembly**.



   - Click **...** to browse.

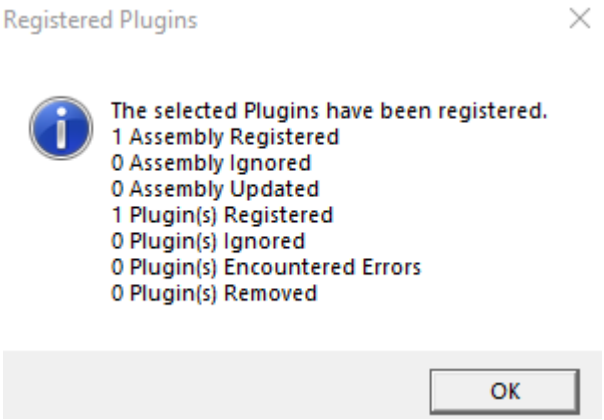


- Browse to the bin/**debug** folder of your plugin project (**ContosoPackageProject**), select the **ContosoPackageProject**.dll file and click **Open**.
  Path: PathToFolder/ContosoPackageProject/ContosoPackageProject/bin/Debug

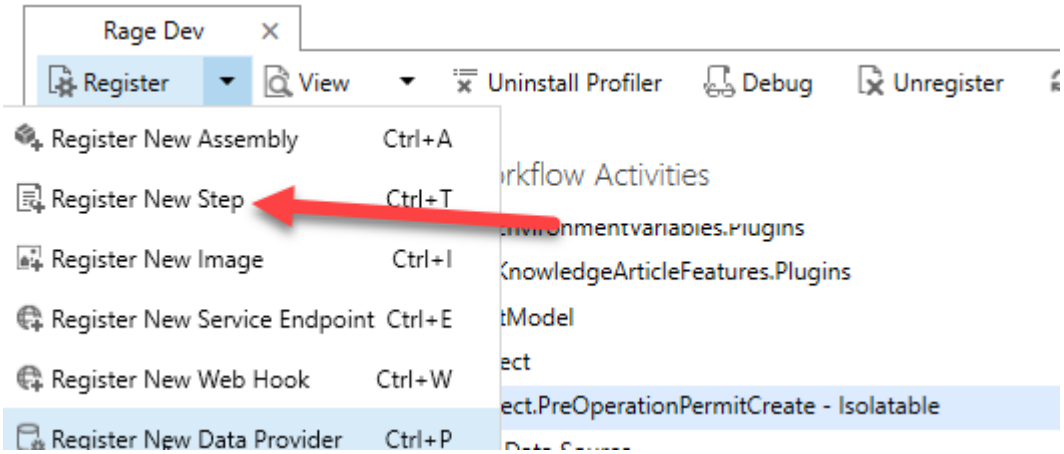


- Click **Register Selected Plugins**.

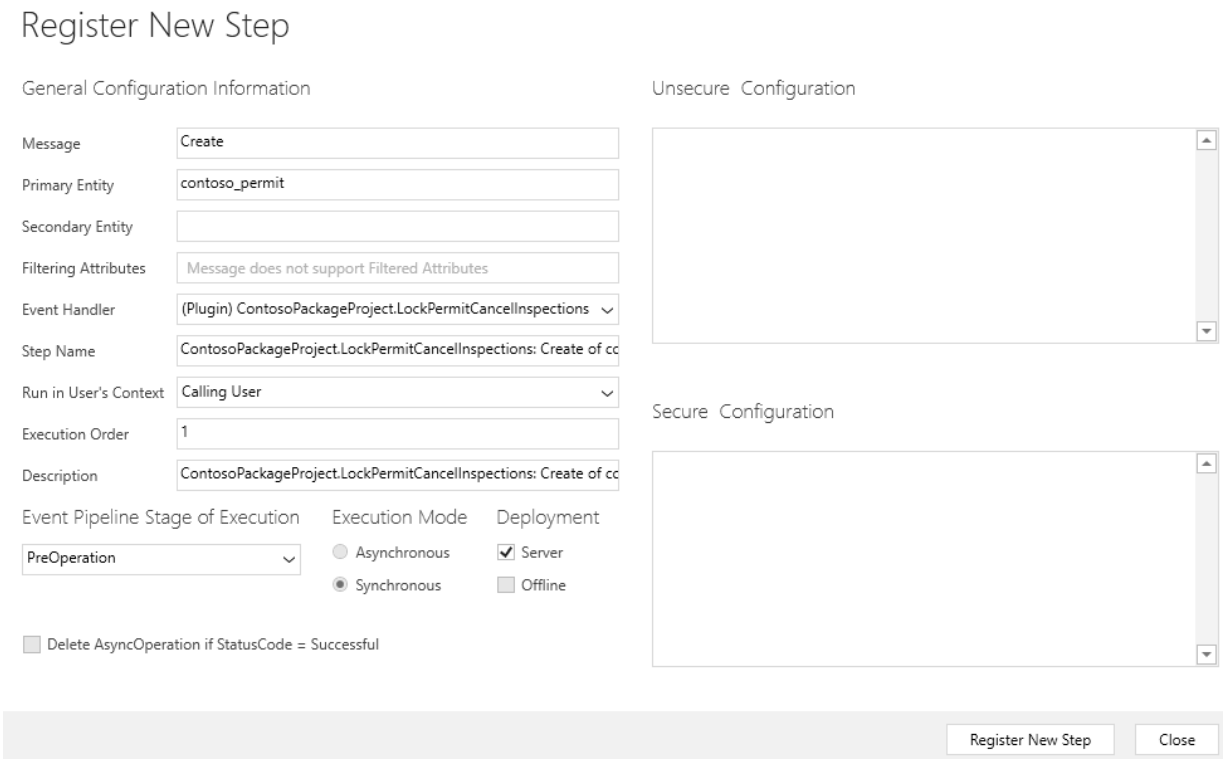
- Click **OK**.



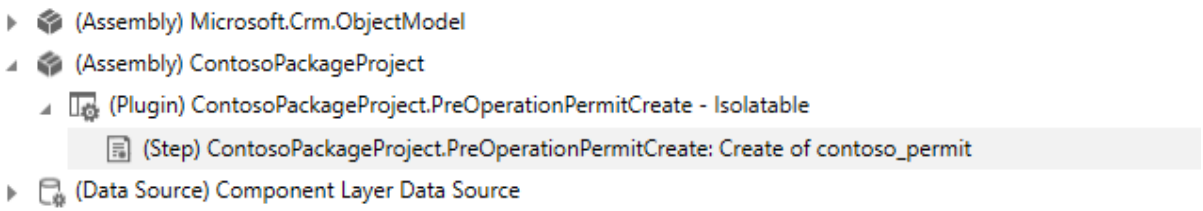
5. Register new step
   - Select the assembly you just registered.
   - Click **Register** and select **Register New Step**.



- Enter **Create** for **Message**.
- Enter **contoso_permit** for **Primary Table**.
- Select **PreOperation** from dropdown for **Event Pipeline Stage of Execution** and click **Register New Step**.



- Step should now be registered in the assembly plugin.



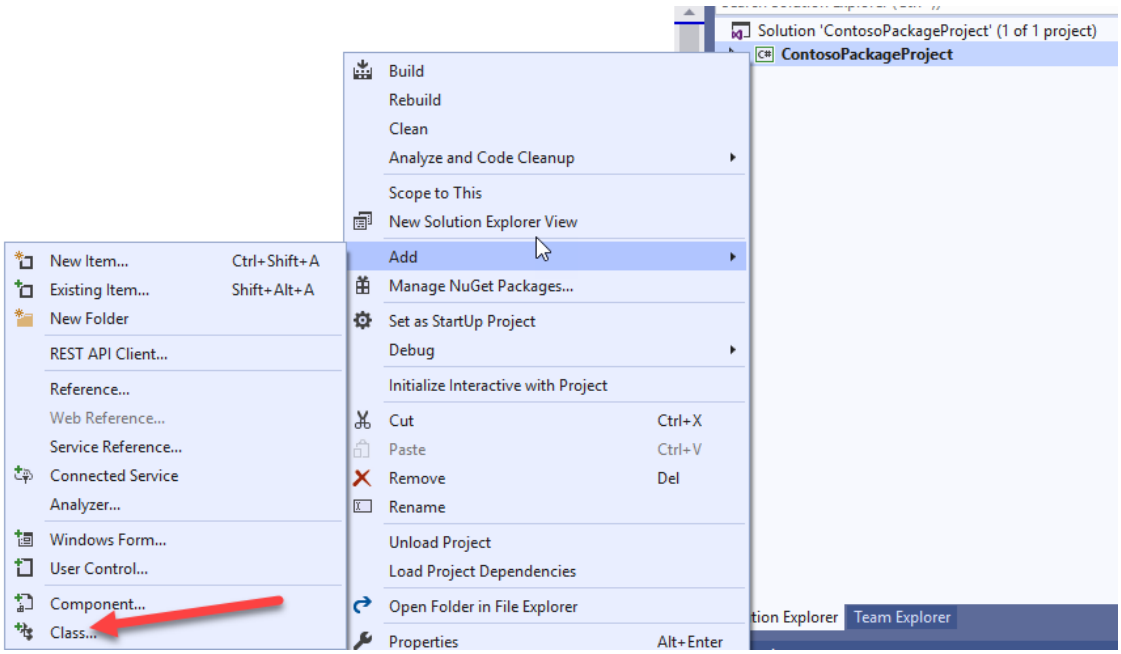
30 Exercise #2: Create Custom Action Plugin

Objective: In this exercise, you will create and register a plugin that will be invoked when the lock permit custom action is used. This plugin will be used to implement the business logic of locking the permit. Specifically, it will update the permit to indicate it is locked and then cancel any pending inspections.

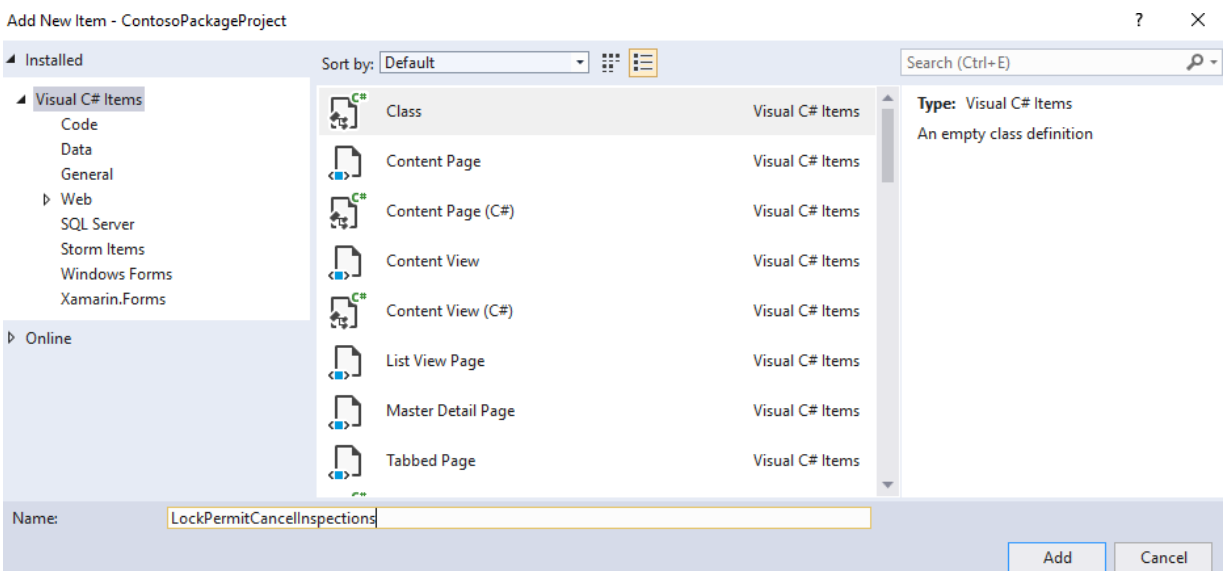
Note: If you did not create the custom action in a prior lab, look in your resources folder for how to add it here before you proceed.

30.1 Task #1: Add a new plugin to the project

1. Add new class to the project and name it **LockPermitCancelInspections**
   - Right on the project and select **Add > Class**.



   - Enter **LockPermitCancelInspections** for **Name** and click **Add**.



2. Add **using statements** to the **LockPermitCancelInspections** class, make the class **public**, and inherit from **PuginBase**
   - Add the using statement below to the **LockPermitCancelInspections** class.

```
using Microsoft.Xrm.Sdk;
using System.Text.RegularExpressions;
using ContosoPackagePoject;
using Microsoft.Xrm.Sdk.Query;
```

   - Make the **LockPermitCancelInspections** public and **inherit** from **PluginBase**.

```
using System.Text;
using System.Threading.Tasks;

using Microsoft.Xrm.Sdk;
using System.Text.RegularExpressions;


namespace ContosoPackageProject
{
    0 references
    public class LockPermitCancelInspections : PluginBase
    {
    }
}
```

3. To override the ExecuteCDSPlugin method and get the reason value from the input parameter.
   - Override the **ExecuteCDSPlugin** method. Add the code below inside the **LockPermitCancelInspections** method.

```
protected override void ExecuteCDSPlugin(LocalPluginContext localcontext)
{
base.ExecuteCDSPlugin(localcontext);
}
```

4. Get the target Table reference, Table, set status reason to lock, and update the permit record.
   - Get the target **Table Reference** and **Table**.

```
var permitEntityRef = localcontext.PluginExecutionContext.InputParameters["Target"] as Entit
Entity permitEntity = new Entity(permitEntityRef.LogicalName, permitEntityRef.Id);
```

   - Add **Trace** message and Set the **Status Reason** to **Lock**. 463270000 is the lock value of the Status Reason option-set and statuscode is the name of the status reason Column.

```
localcontext.Trace("Updating Permit Id : " + permitEntityRef.Id);
permitEntity["statuscode"] = new OptionSetValue(463270000);
```

   - Update the **Permit** record and add **Trace** message.

```
localcontext.OrganizationService.Update(permitEntity);
localcontext.Trace("Updated Permit Id " + permitEntityRef.Id);
```

```
5 references
protected override void ExecuteCDSPlugin(LocalPluginContext localcontext)
{
    base.ExecuteCDSPlugin(localcontext);
    var permitEntityRef = localcontext.PluginExecutionContext.InputParameters["Target"] as EntityReference;
    Entity permitEntity = new Entity(permitEntityRef.LogicalName, permitEntityRef.Id);
    localcontext.Trace("Updating Permit Id : " + permitEntityRef.Id);
    permitEntity["statuscode"] = new OptionSetValue(463270000);
    localcontext.OrganizationService.Update(permitEntity);
    localcontext.Trace("Updated Permit Id " + permitEntityRef.Id);

}
```

30.2 Task #2: Get Related Inspections and Cancel

1. Create query and condition expressions.
   - Create the **QueryExpression**. Add the code below to the **ExecuteCDSPlugin** method.

```
QueryExpression qe = new QueryExpression();
qe.EntityName = "contoso_inspection";
qe.ColumnSet = new ColumnSet("statuscode");
```

   - Create the **ConditionExpression**.

```
ConditionExpression condition = new ConditionExpression();
condition.Operator = ConditionOperator.Equal;
condition.AttributeName = "contoso_permit";
condition.Values.Add(permitEntityRef.Id);
```

- Set the **Criteria** of the query.

```
qe.Criteria = new FilterExpression(LogicalOperator.And);
```

- Add the **ConditionExpression** to the **Criteria** of the **QueryExpression**.

```
qe.Criteria.Conditions.Add(condition);
```

```
    localcontext.Trace("Updated Permit Id " + permitEntityRef.Id);

    QueryExpression qe = new QueryExpression();
    qe.EntityName = "contoso_inspection";
    qe.ColumnSet = new ColumnSet("statuscode");

    ConditionExpression condition = new ConditionExpression();
    condition.Operator = ConditionOperator.Equal;
    condition.AttributeName = "contoso_permit";
    condition.Values.Add(permitEntityRef.Id);

    qe.Criteria = new FilterExpression(LogicalOperator.And);
    qe.Criteria.Conditions.Add(condition);

}
```

2. Retrieve the inspections and iterate through the returned Tables.

- Retrieve the **Inspections** and add **Trace** messages.

```
localcontext.Trace("Retrieving inspections for Permit Id " + permitEntityRef.Id);
var inspectionsResult = localcontext.OrganizationService.RetrieveMultiple(qe);
localcontext.Trace("Retrievied " + inspectionsResult.TotalRecordCount + " inspection records
```

- Create a **variable** that will keep track of the canceled **Inspections** count and Iterate through the returned Tables.

```
int canceledInspectionsCount = 0;
foreach (var inspection in inspectionsResult.Entities)
{

}
```

3. Retrieve the selected status reason option and check if it is set to new request or pending.

- Get the currently selected value of the **Status Reason** option-set. Add the code below inside the **foreach** loop.

```
    var currentValue = inspection.GetAttributeValue<OptionSetValue>("statuscode");
```

- Check if the selected option is **New Request** or **Pending** and increment the count. 1 is the value of the New Request option and 463270000 id the value of the Pending option. This should be placed inside the foreach loop.

```
if (currentValue.Value == 1 || currentValue.Value == 463270000)
{
canceledInspectionsCount++;
}
```

4. Cancel the inspections that are pending or new request

- Set the **Status Reason** selected value to **Canceled**. Add the code below inside the if statement inside the foreach loop. Make sure that 463270003 is the value for **Canceled Status Reason** in the

Inspections Table. If this differs, please update the value with actual value for **Canceled Status Reason**.

```
inspection["statuscode"] = new OptionSetValue(463270003);
```

- Update the **Inspection** and add **Trace** messages.

```
localcontext.Trace("Canceling inspection Id : " + inspection.Id);
localcontext.OrganizationService.Update(inspection);
localcontext.Trace("Canceled inspection Id : " + inspection.Id);
```

```
qe.Criteria.Conditions.Add(condition);

localcontext.Trace("Retrieving inspections for Permit Id " + permitEntityRef.Id);
var inspectionsResult = localcontext.OrganizationService.RetrieveMultiple(qe);
localcontext.Trace("Retrieved " + inspectionsResult.TotalRecordCount + " inspection records");

int canceledInspectionsCount = 0;
foreach (var inspection in inspectionsResult.Entities)
{
    var currentValue = inspection.GetAttributeValue<OptionSetValue>("statuscode");

    if (currentValue.Value == 1 || currentValue.Value == 463270000)
    {
        canceledInspectionsCount++;

        inspection["statuscode"] = new OptionSetValue(463270003);
        localcontext.Trace("Canceling inspection Id : " + inspection.Id);
        localcontext.OrganizationService.Update(inspection);
        localcontext.Trace("Canceled inspection Id : " + inspection.Id);
    }
}
```

30.3 Task #3: Set Output Parameter and Create Note Record

1. Check if at least one Inspection was canceled and CanceledInspectionsCount output Parameter.
   - Check if at least one **Inspection** was canceled. Add the code below after the **foreach** loop.

```
if (canceledInspectionsCount > 0)
{

}
```

   - Set the **CanceledInspectionsCount** output parameter. Add the code below inside the if statement outside the foreach loop.

```
localcontext.PluginExecutionContext.OutputParameters["CanceledInspectionsCount"] = canceledI
```

```
    }
}

if (canceledInspectionsCount > 0)
{
    localcontext.PluginExecutionContext.OutputParameters["CanceledInspectionsCount"] = canceledInspectionsCount + " Inspections were canceled";
}
```

2. Check if the Input Parameters contain Reason and Create the Note record.
   - Check if **Reason** contains in the **InputParameters**. Add the code below after the last if statement.

```
if (localcontext.PluginExecutionContext.InputParameters.ContainsKey("Reason"))
{

}
```

   - Build the **Note** record and add **Trace Message**. Add the code below inside the if statement.

```
localcontext.Trace("building a note reocord");
Entity note = new Entity("annotation");
note["subject"] = "Permit Locked";
note["notetext"] = "Reason for locking this permit: " + localcontext.PluginExecutionContext.
```

```
note["objectid"] = permitEntityRef;
note["objecttypecode"] = permitEntityRef.LogicalName;
```

- Add Trace Message and create the Note record.

```
localcontext.Trace("Creating a note reocord");
var createdNoteId = localcontext.OrganizationService.Create(note);
```

- Check if the Note record was created and add Trace Message.

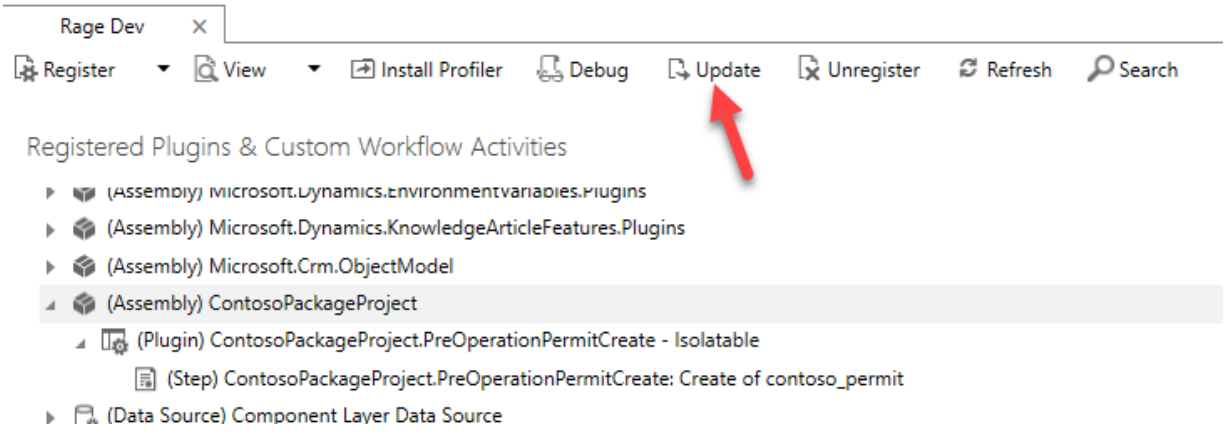
```
if (createdNoteId != Guid.Empty)
localcontext.Trace("Note record was created");
```

```
if (canceledInspectionsCount > 0)
{
    localcontext.PluginExecutionContext.OutputParameters["CanceledInspectionsCount"] = canceledInspectionsCount + " Inspections were canceled";
}

if (localcontext.PluginExecutionContext.InputParameters.ContainsKey("Reason"))
{
    localcontext.Trace("building a note reocord");
    Entity note = new Entity("annotation");
    note["subject"] = "Permit Locked";
    note["notetext"] = "Reason for locking this permit: " + localcontext.PluginExecutionContext.InputParameters["Reason"];
    note["objectid"] = permitEntityRef;
    note["objecttypecode"] = permitEntityRef.LogicalName;

    localcontext.Trace("Creating a note reocord");
    var createdNoteId = localcontext.OrganizationService.Create(note);
    if (createdNoteId != Guid.Empty)
        localcontext.Trace("Note record was created");
}
}
```

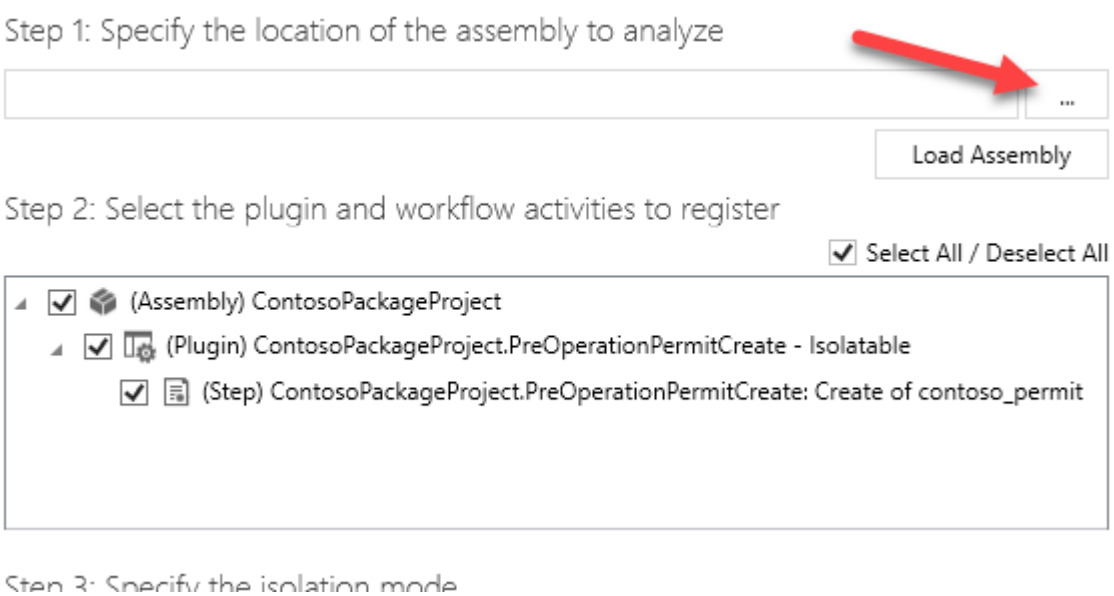
3. Build plugin by right clicking on the project and select **Build** and make sure the build succeeds.

30.4 Task #4: Deploy Plugin

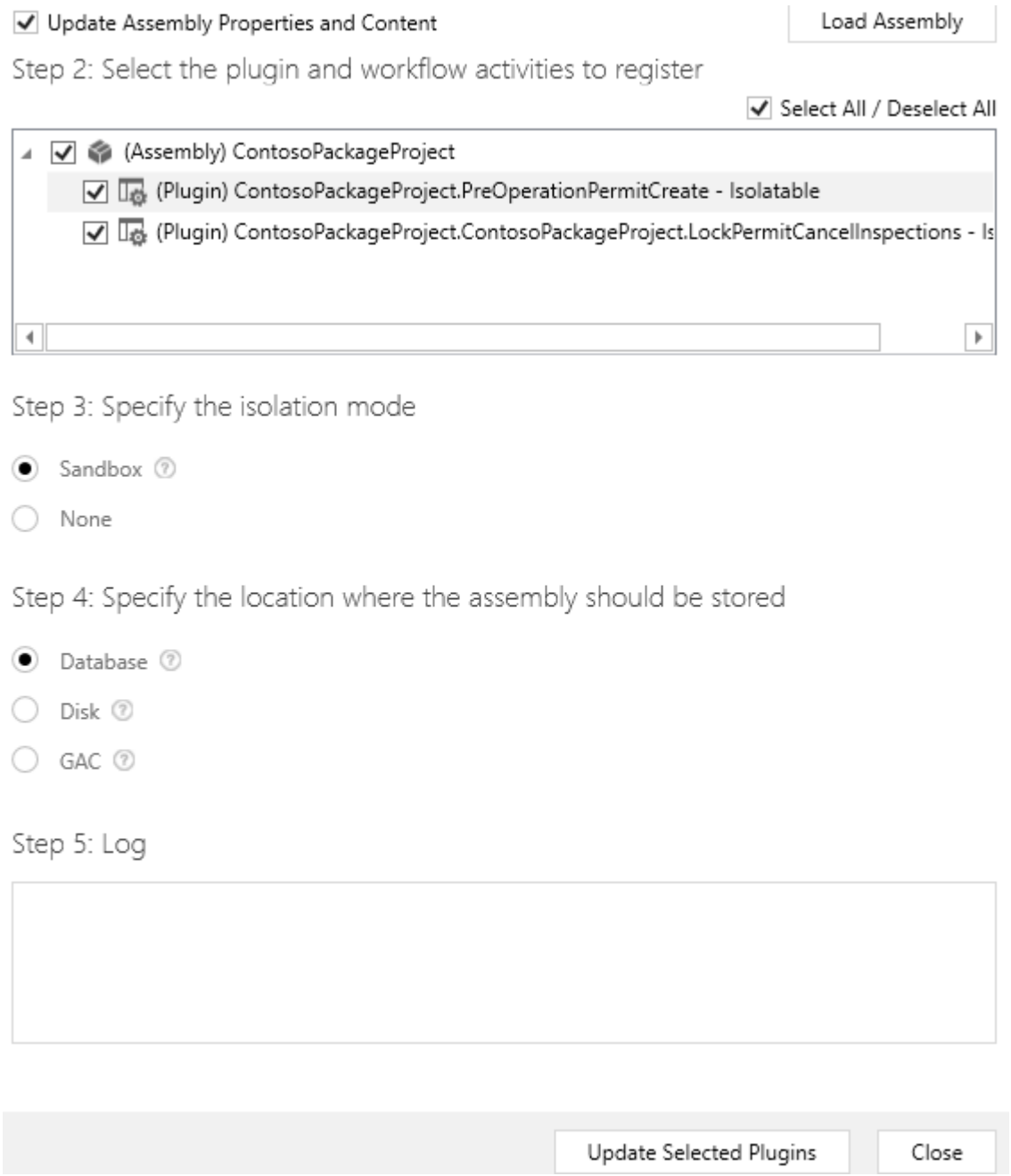
1. If you do not have the plugin registration tool running already, follow instructions in Exercise #1, Task #2 to run the tool and connect to the organization.
2. Update the assembly
   - Select **ContosoPackageProject** and click **Update**.



   - Click … to **Browse**.

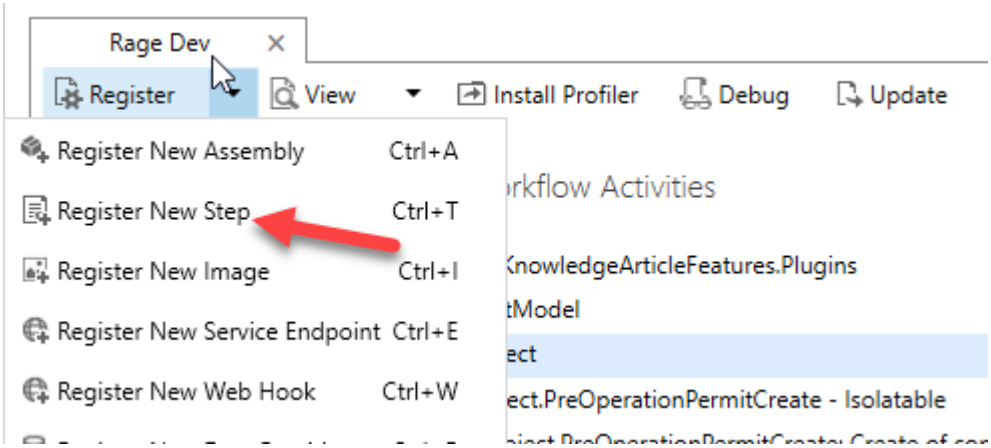


- Browse to the **debug** folder of your plugin project, select the **ContosoPackageProject**.dll file and click **Open**.
- Check **Select All** checkbox and click **Update Selected Plugins**.

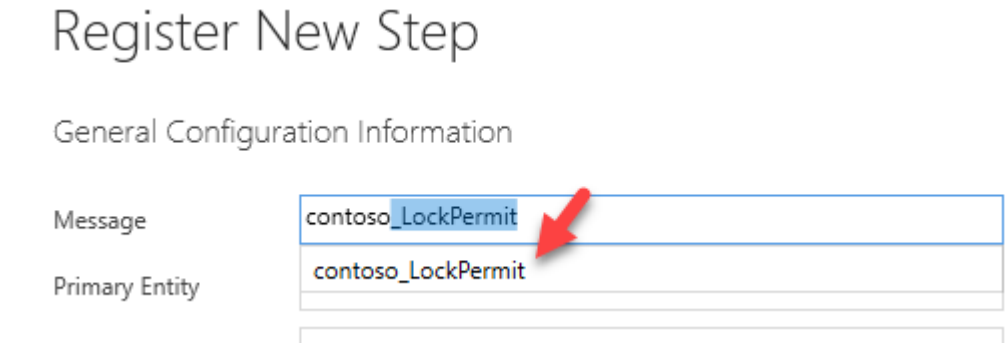


- Click **OK**.

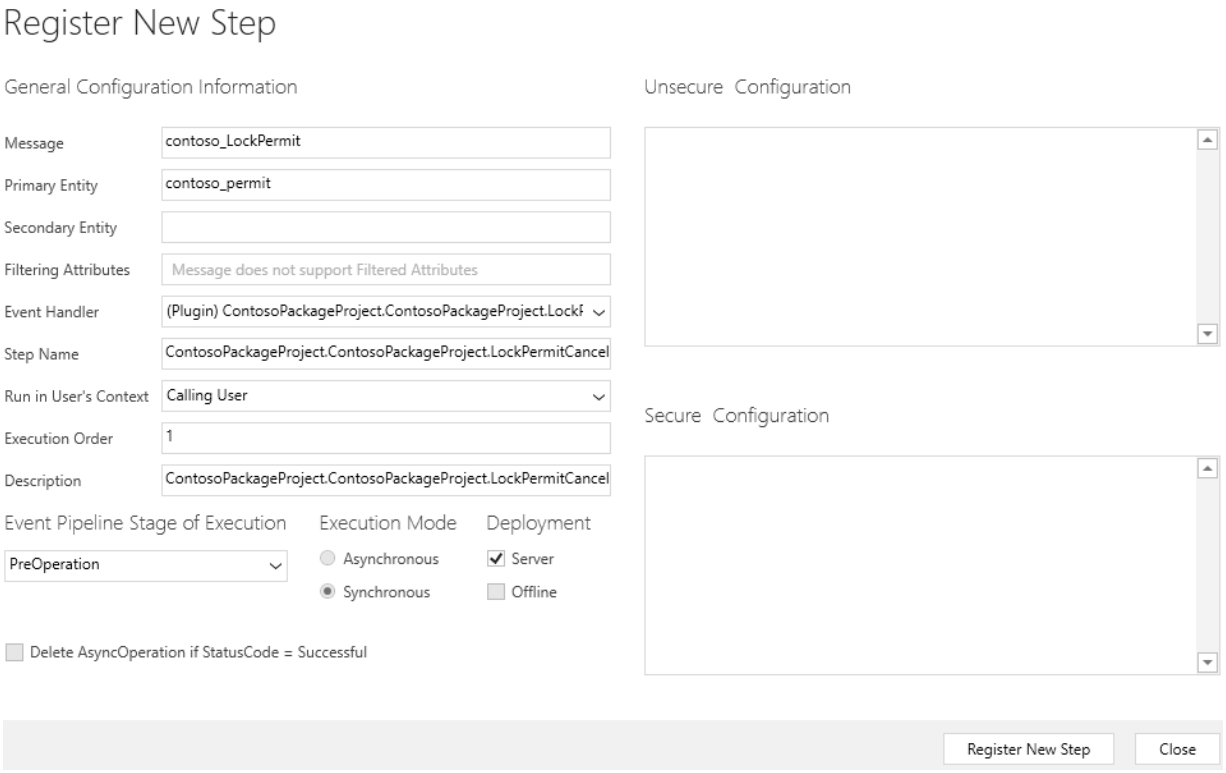
3. Register new step
   - Select the assembly (ContosoPackageProject.LockPermitCancelInspections).
   - Click **Register** and select **Register New Step**.



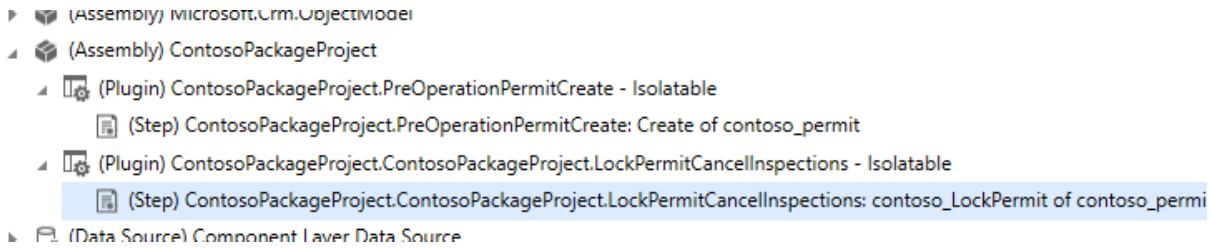
- Enter **contoso** in the **Message** textbox and select **contoso_LockPermit**.



- Enter **contoso_permit** for **Primary Table**.
- Make sure that Event Handler is selected for **LockPermitCancelInspections** plugin.
- Select **PreOperation** from dropdown for **Event Pipeline Stage of Execution** and click **Register New Step**.



- Step should now be registered in the assembly.



31 Exercise #3: Test Plugins

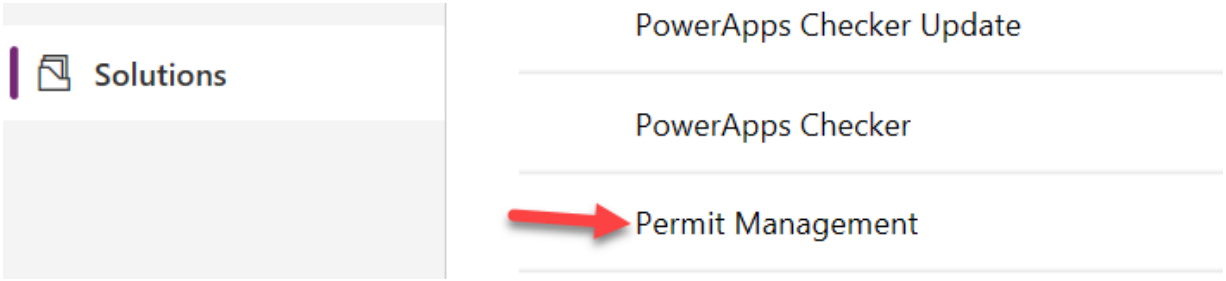
Objective: In this exercise, you will test the plugins you created.

31.1 Task #1: Test Lock Plugin

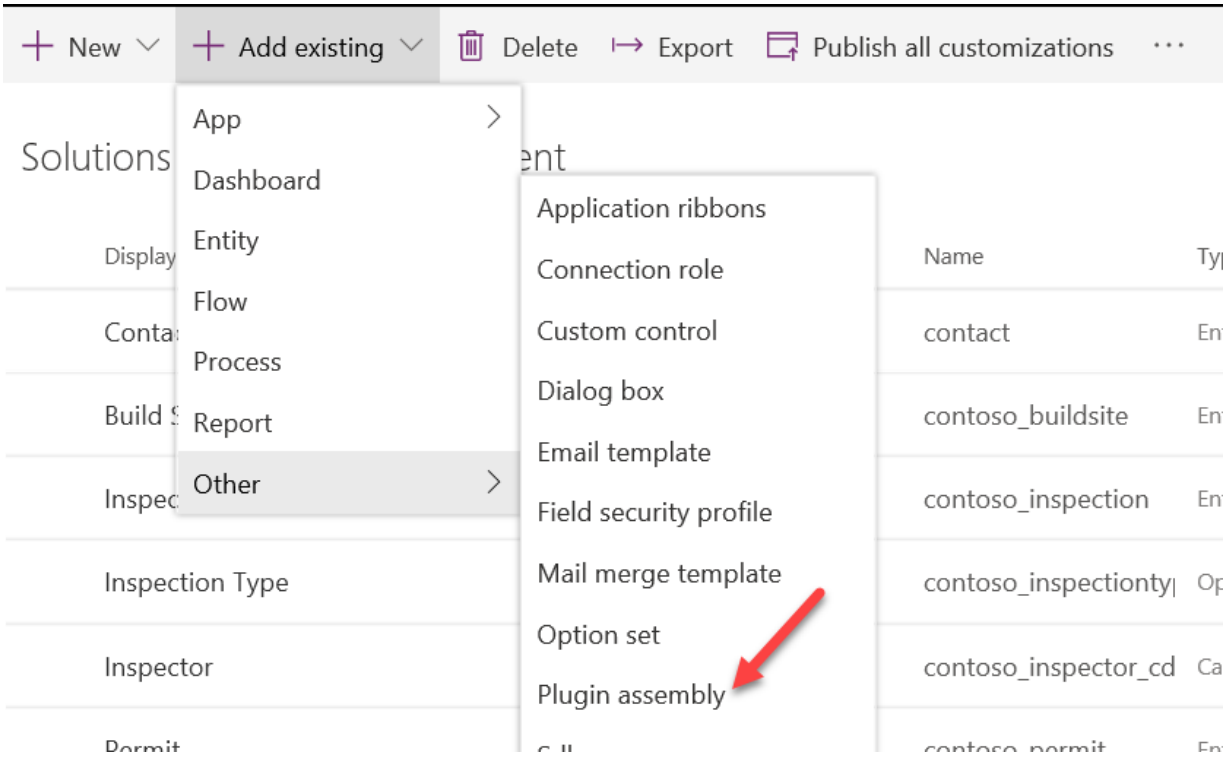
1. Add Plugin Assembly and Plugin - step to the Permit Management solution

Note: This is only required to be done after deploying it first time to ensure that when the project solution is moved to the production environments, the plugins and the registered steps are included.

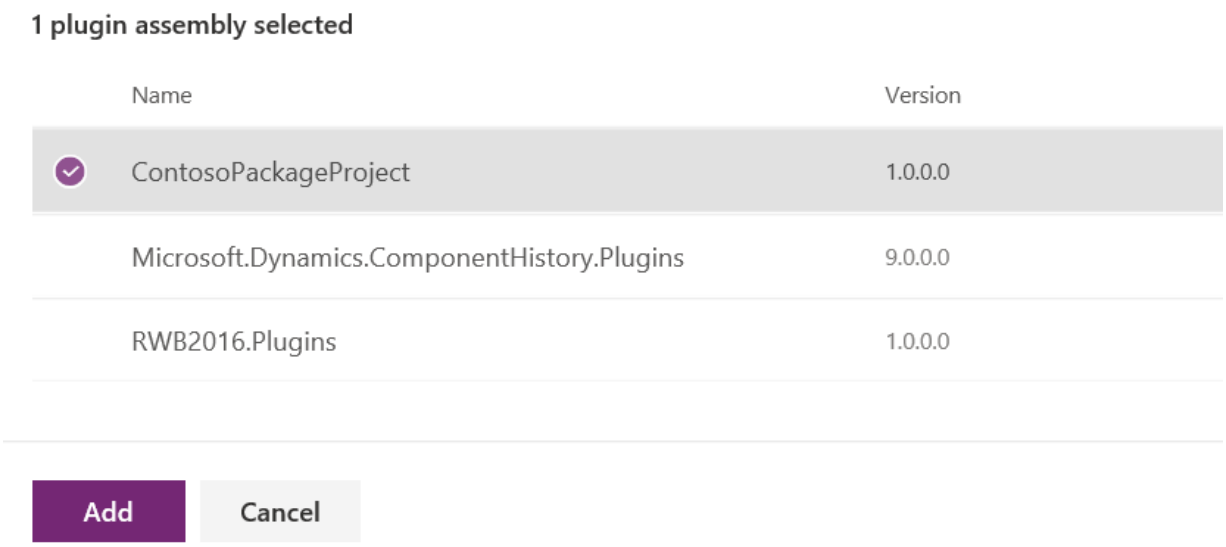
- Sign in to Power Apps maker portal and make sure you have the **Dev** environment selected.
- Select **Solution** and click to open the **Permit Management** solution.



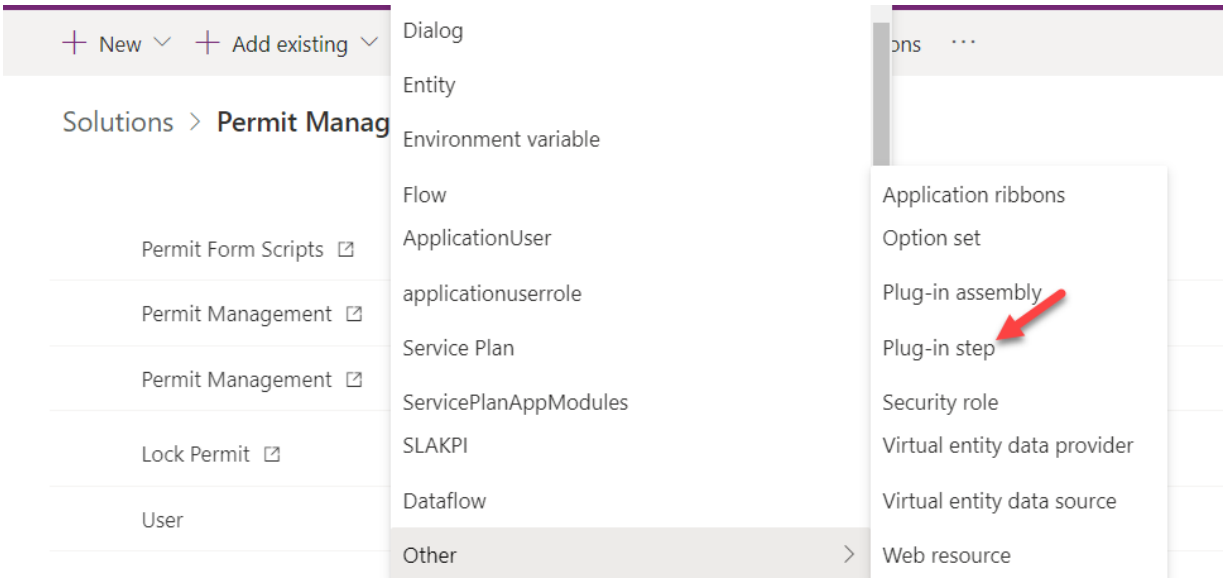
- Click **Add Existing | Other | Plugin Assembly**.



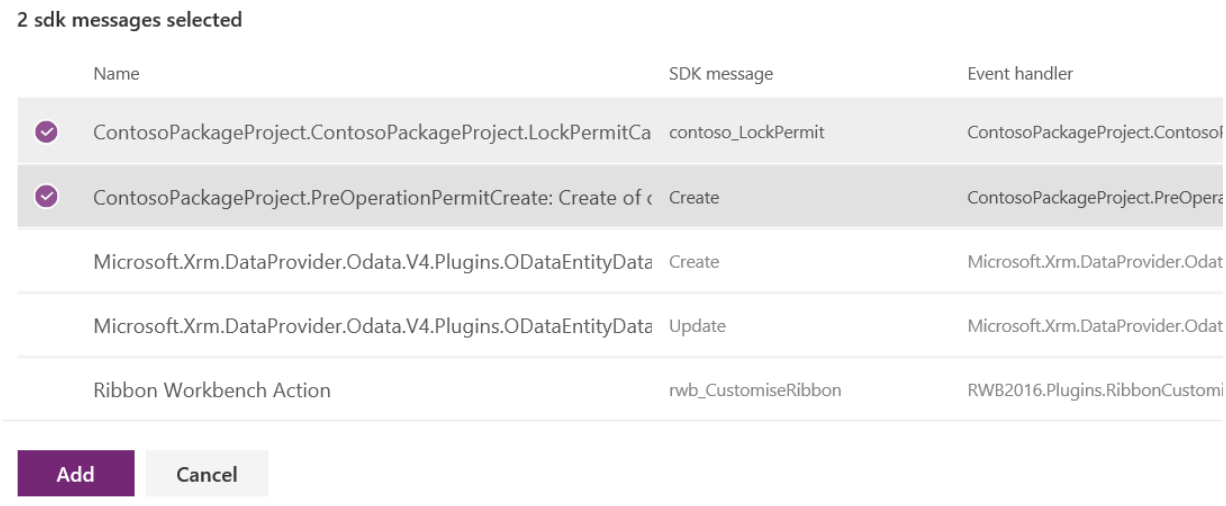
- Select **ContosoPackageProject** and click **Add**.



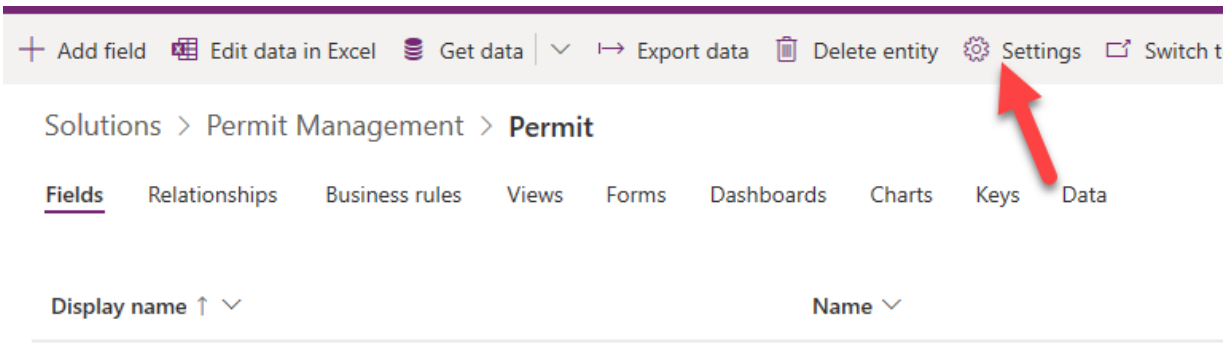
- Click **Add Existing | Other | Plug-in step**.



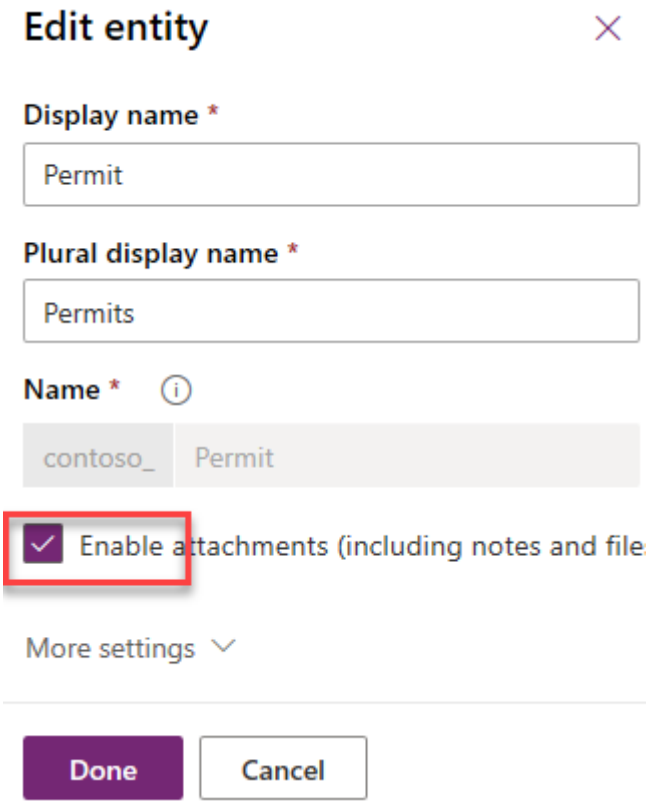
- Select both SDK Messages you created and click **Add**.



- Now, open the **Permit** Table and click **Settings**.

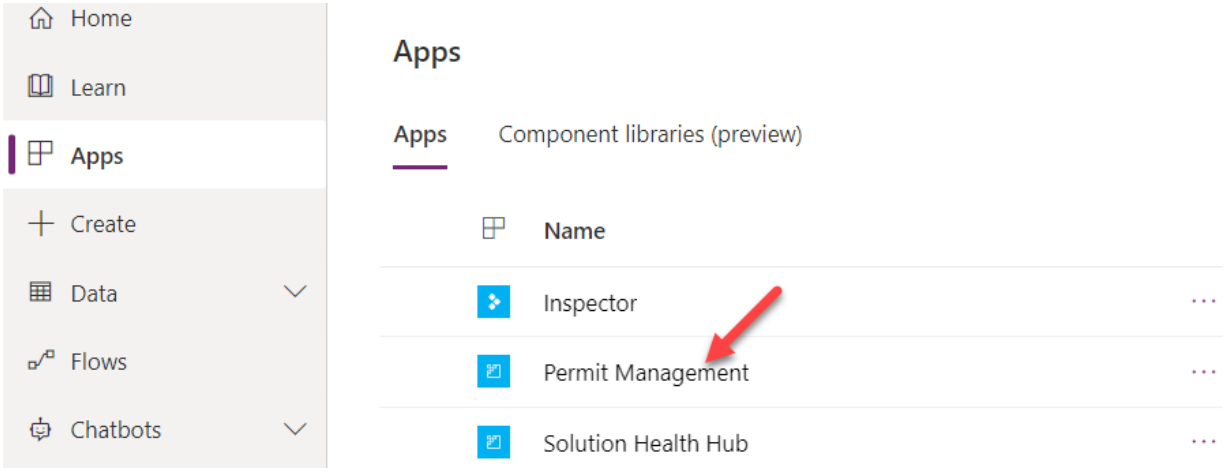


- Check **Enable Attachments and Notes,** and then click **Done**.

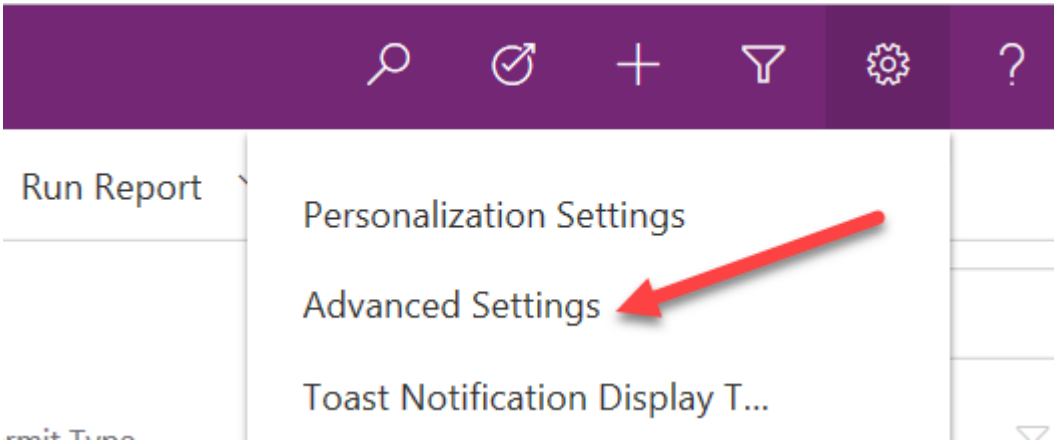


- Click **Okay**.
- Finally, select **Save Table**.
- Select **Solutions** and click **Publish All Customizations**.

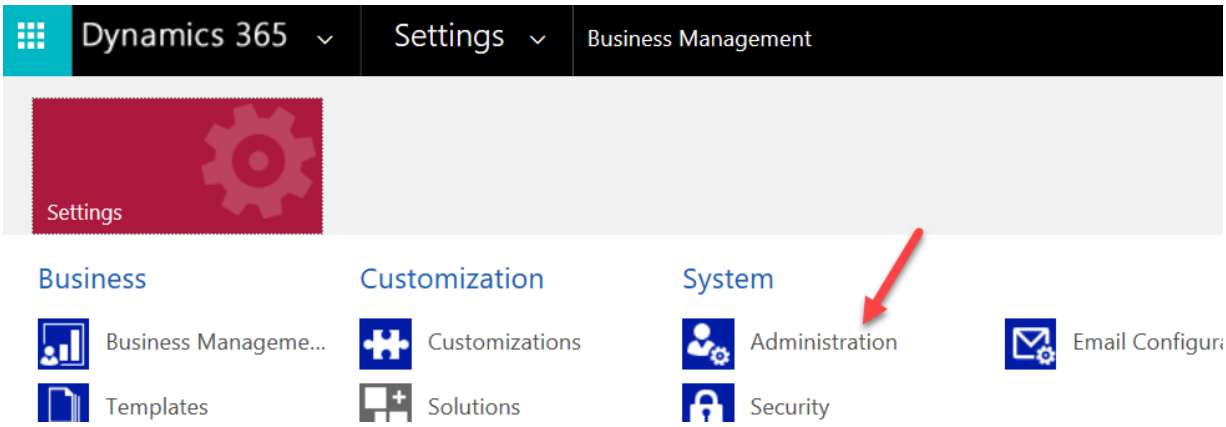
2. Start the Permit Management application and enable Plugin Tracing
   - Select **Apps**.
   - Click to open the **Permit Management** application.



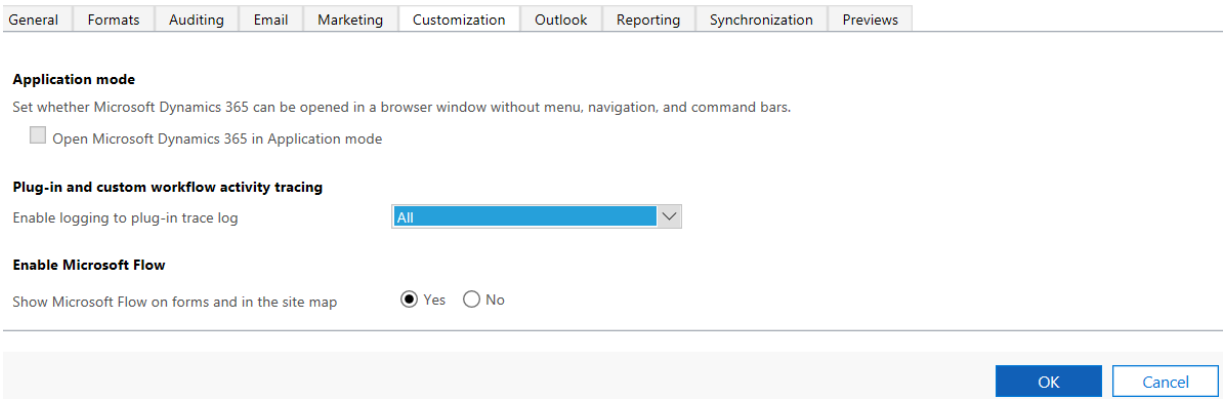
- Click **Settings** and select **Advanced Settings**.



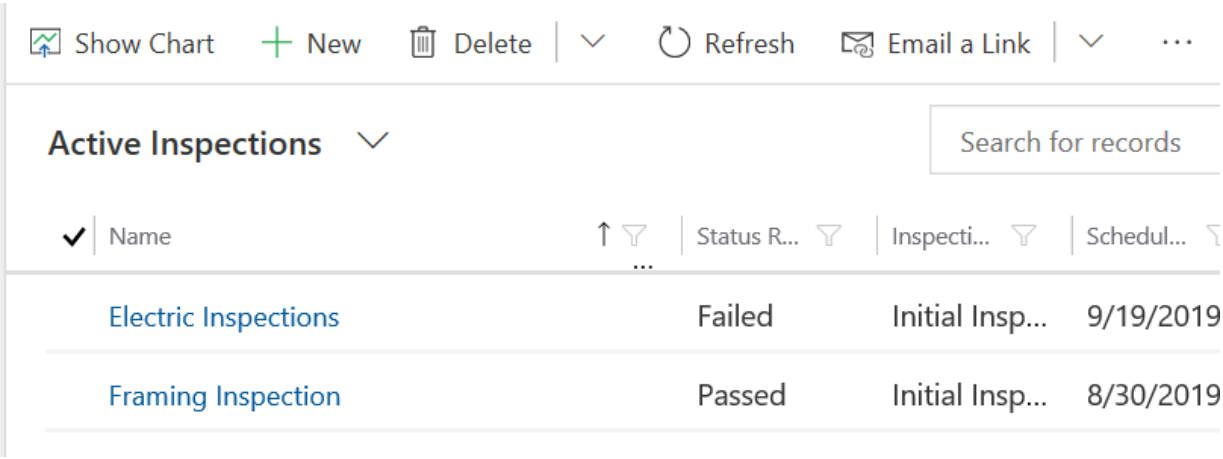
- Click **Settings** and select **Administration**.



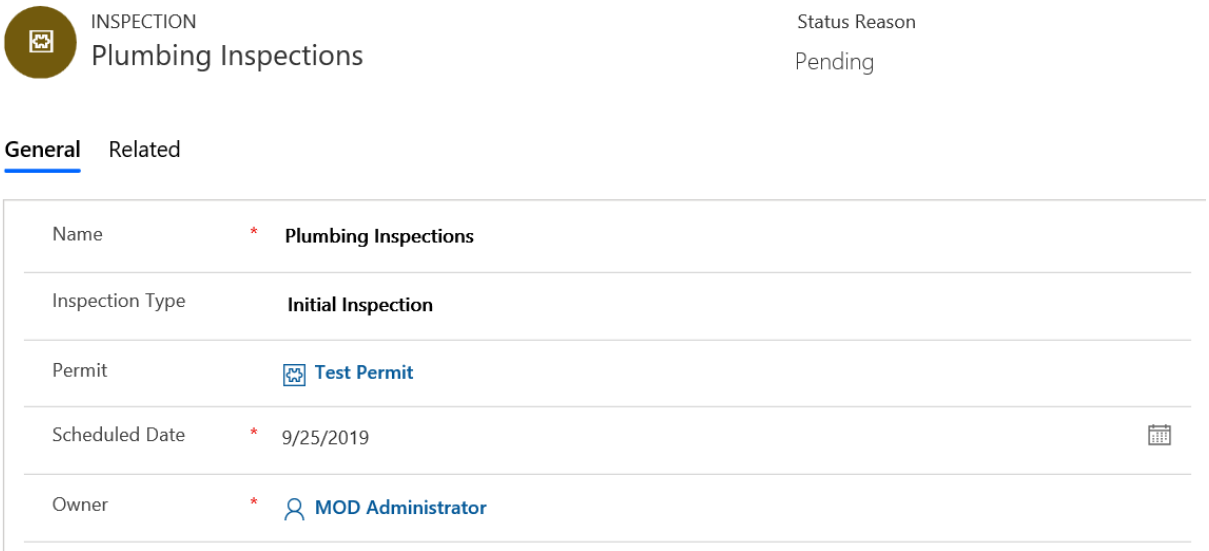
- Click **System Settings**.
- Select **Customization** tab.
- Set Enable Plugin Logging to Plugin Trace Log to **All** and click **OK**.



3. Create test record
   - Go back to the **Permit Management** application.
   - Select **Inspections**.
   - You should have two inspections one **Failed** and one **Passed**. If not, open them and update the records.
   - Click **New**.

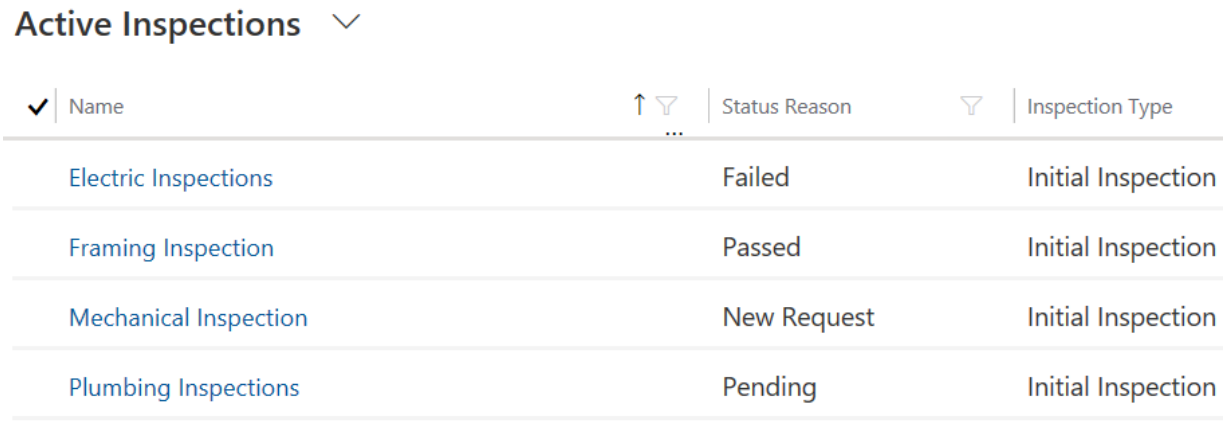


   - Enter **Plumbing Inspection** for **Name**, select **Initial Inspection** for **Type**, select a permit, provide **Schedule Data**, select **Pending** for **Status Reason**, and click **Save**.

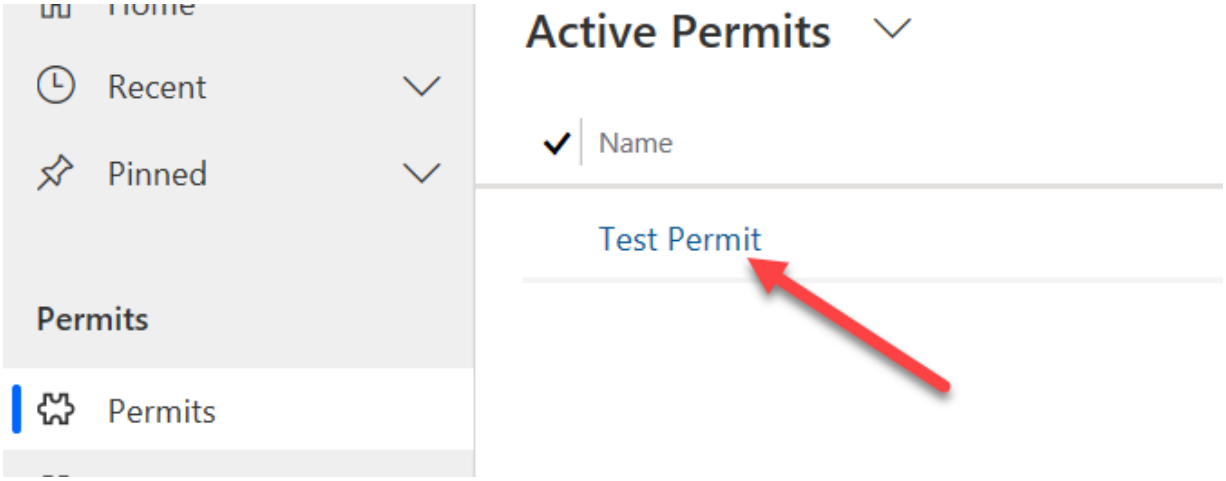


   - Click **New** again.
   - Enter **Mechanical Inspection** for **Name**, select **Initial Inspection** for **Type**, select a permit, provide **Schedule Date**, select **New Request** for **Status Reason**, and click **Save**.

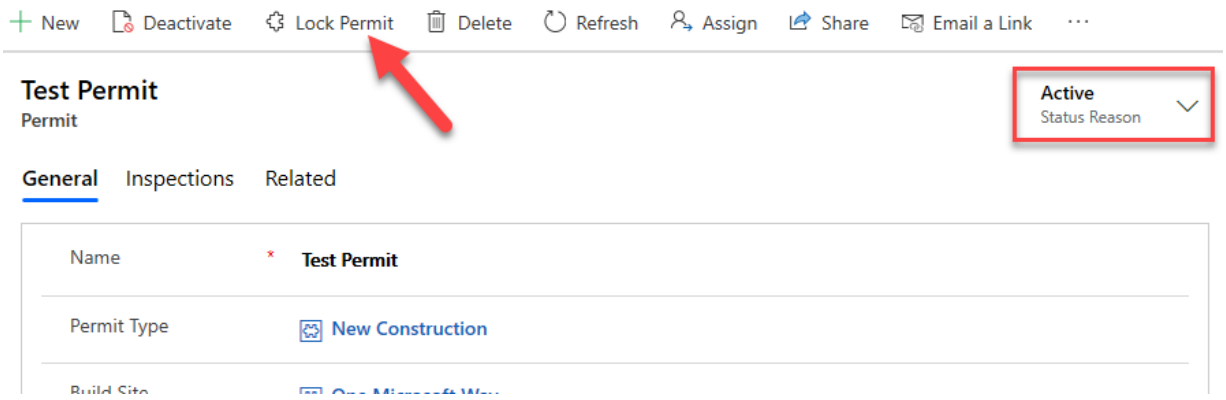
4. Lock Permit.
    - Select **Inspections**.
    - Make sure you have four inspection records and with various Status Reason value.



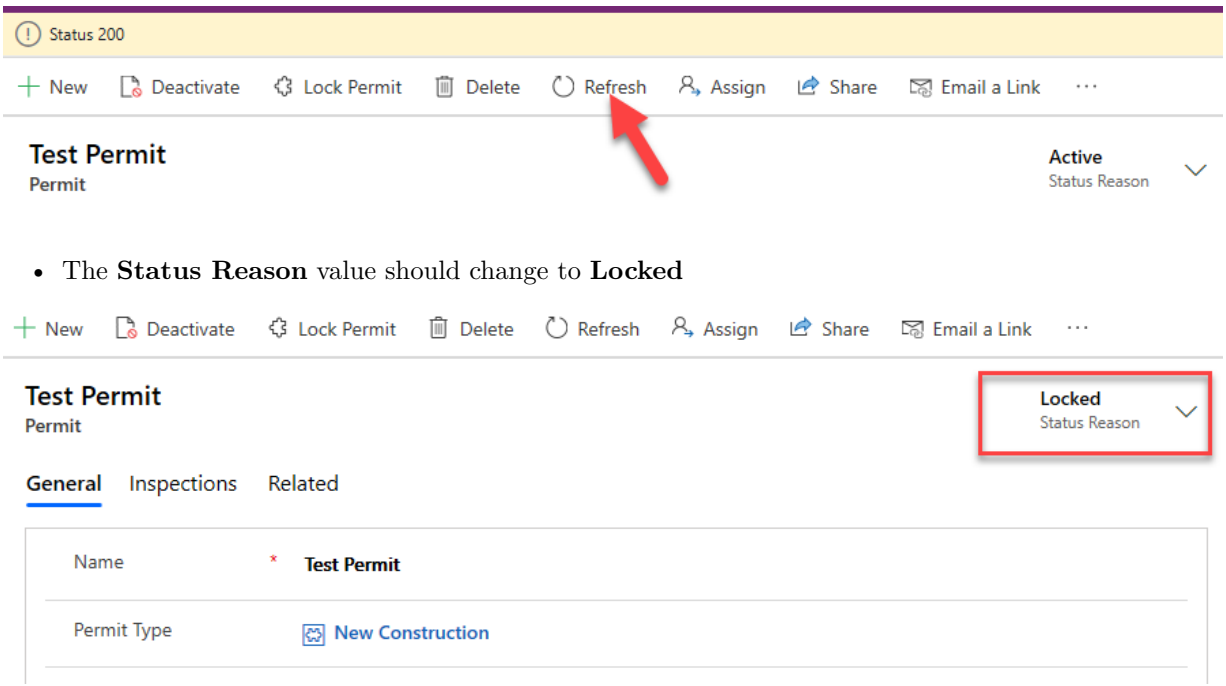
    - Select **Permits**.
    - Click to open the **Test Permit**.



    - Make sure the Status Reason is set to Active and click **Lock Permit**.



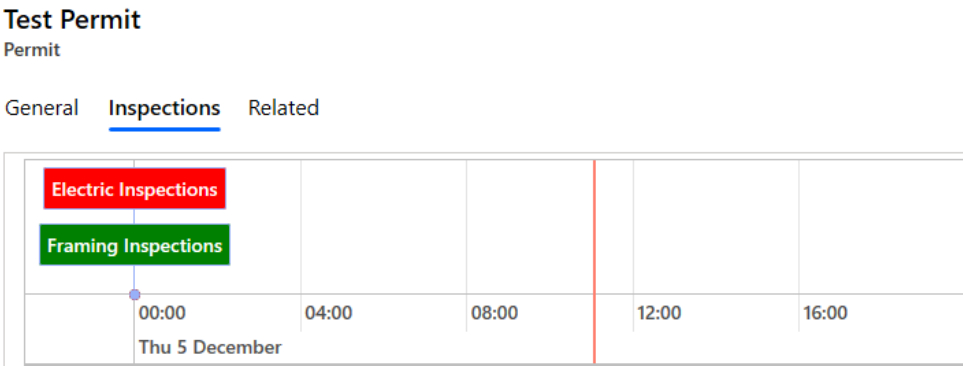
    - The Custom Action should run. Click **Refresh**.



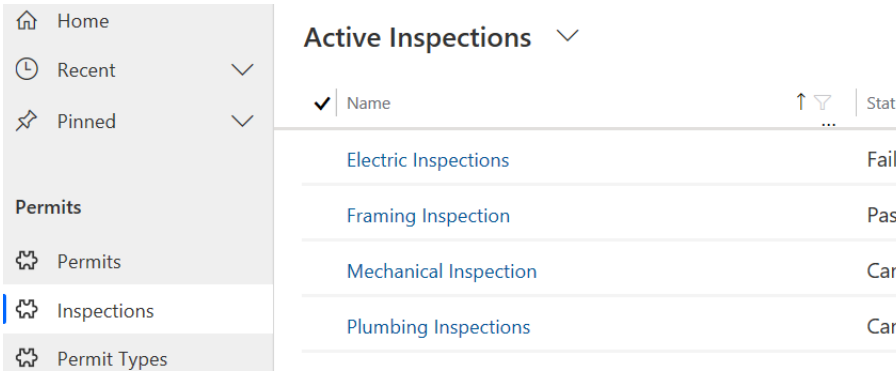
- The **Status Reason** value should change to **Locked**

5. Check if the Pending and New Request Inspections get canceled
   - Select Inspections.
   - You should now have two canceled inspections.

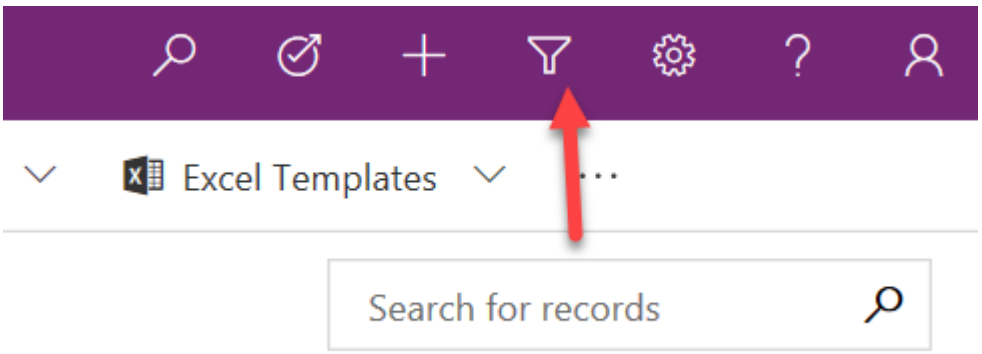
If you completed the PCF module:



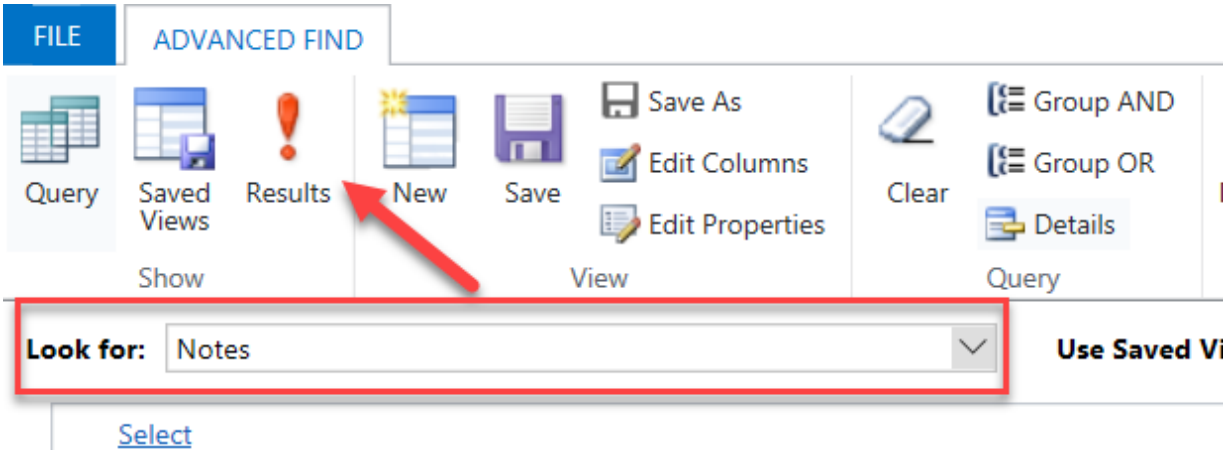
If you did not complete the PCF module:



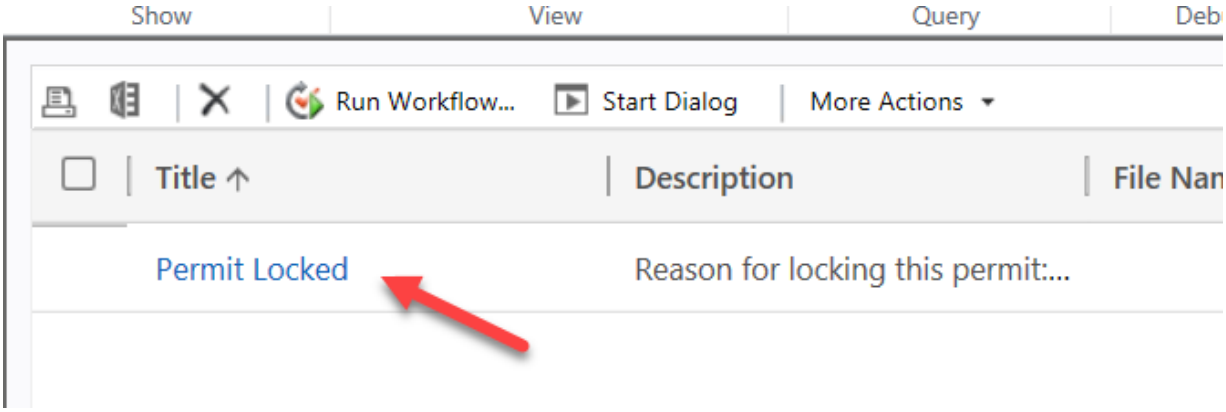
6. Check if the Note record was created.
   - Click **Advanced Find**.



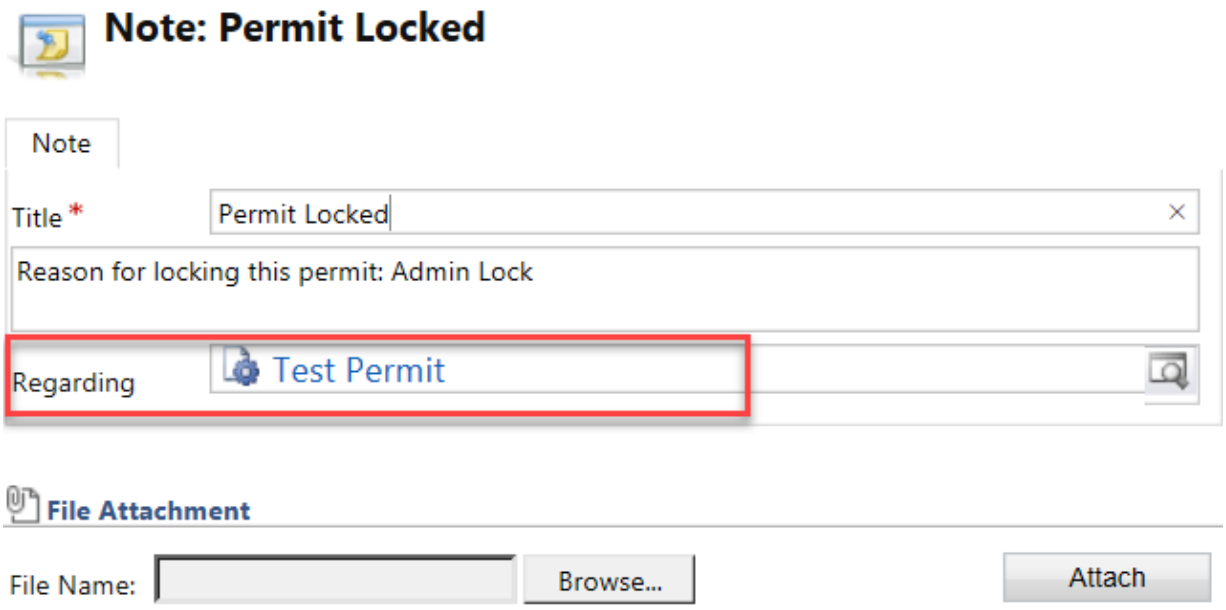
- Select **Notes** and click **Results**.



- You should at least one **Note** record. Click to open the **Note** record.



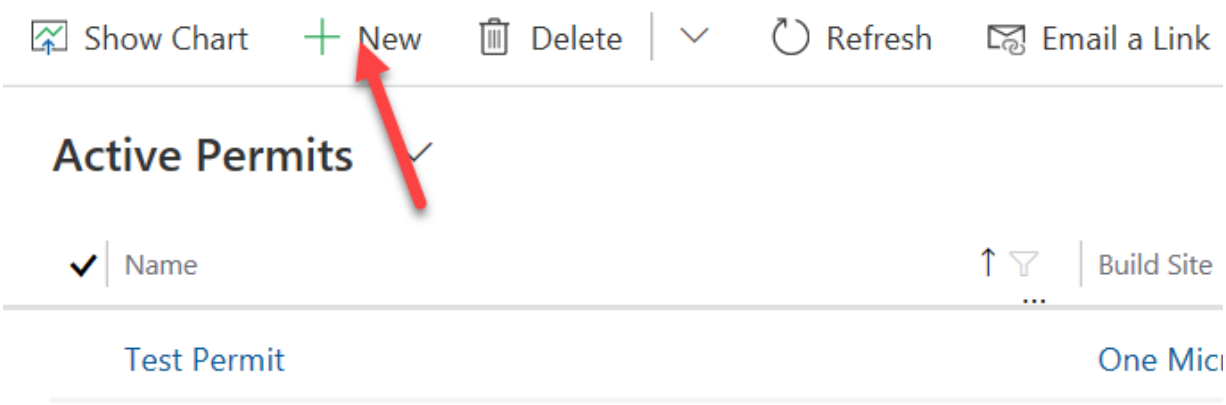
- The Regarding Column should be set to the Permit you locked.



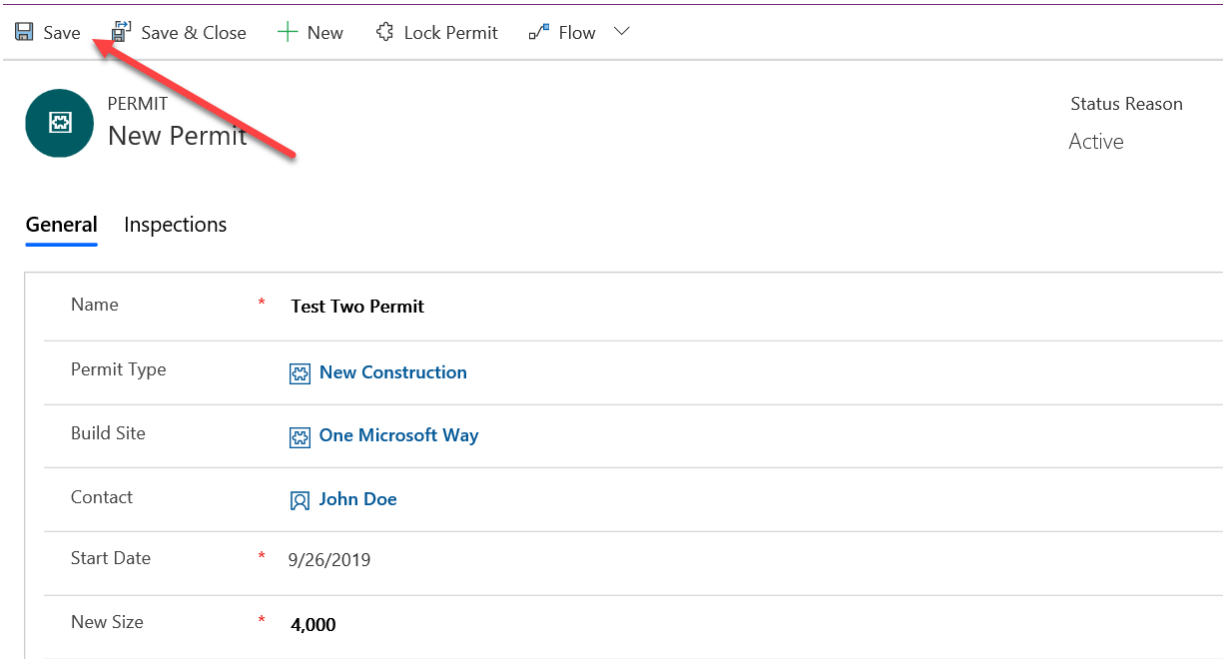
- Close the **Note** record. Close **Advanced Find**.

31.2 Task #2: Test Restrict New Permit Creation Plugin

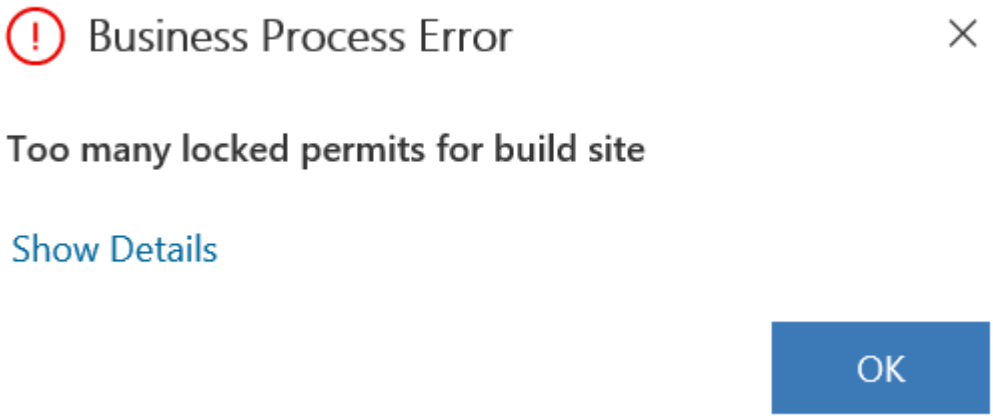
1. Try to create new Permit record for the One Microsoft Way Build Site
   - Select **Permits**.
   - Click **New**.



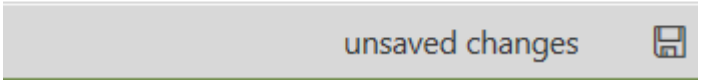
   - Provide the information below and click **Save**.



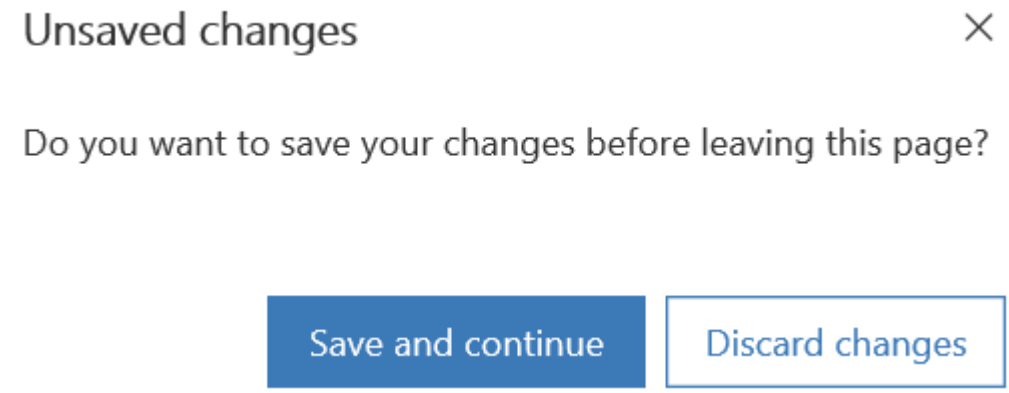
- You should get the error below. Click **OK**.



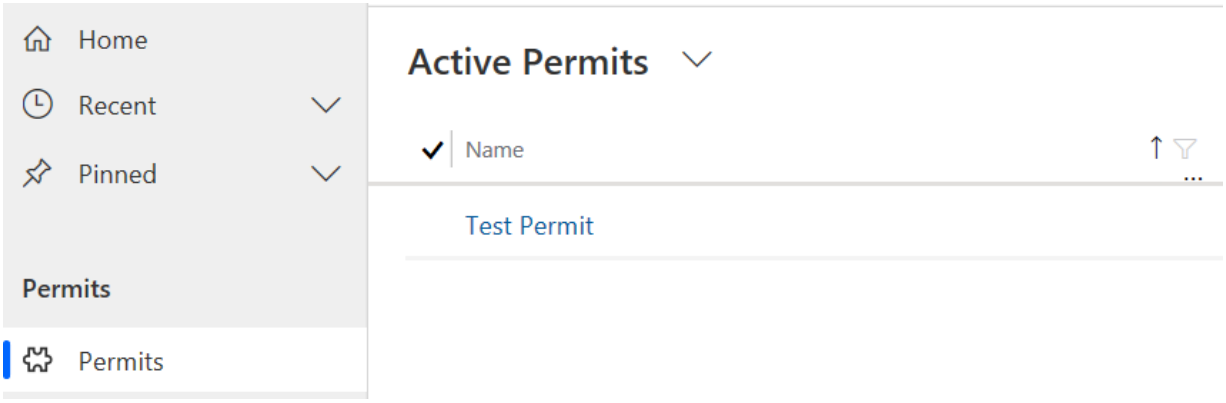
- The record should not get created.



- Select **Permits**.
- Click **Discard**.



- You should have only one Permit record.

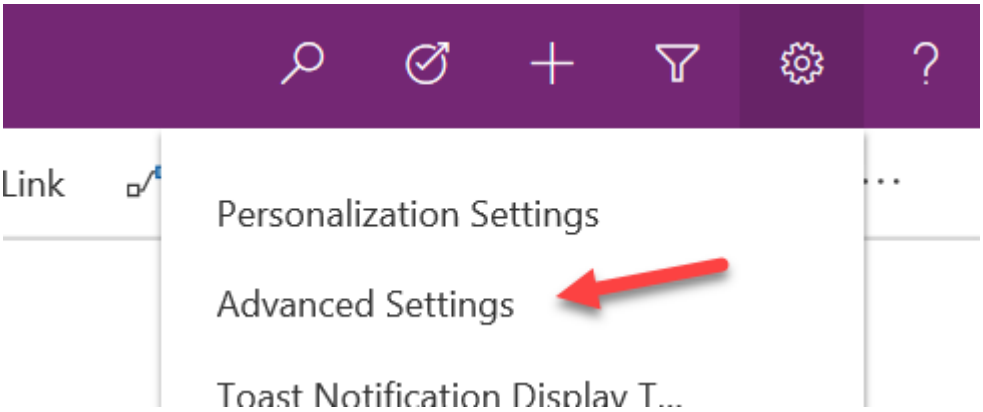


32 Exercise #4: Plugin Trace Log and Debugging

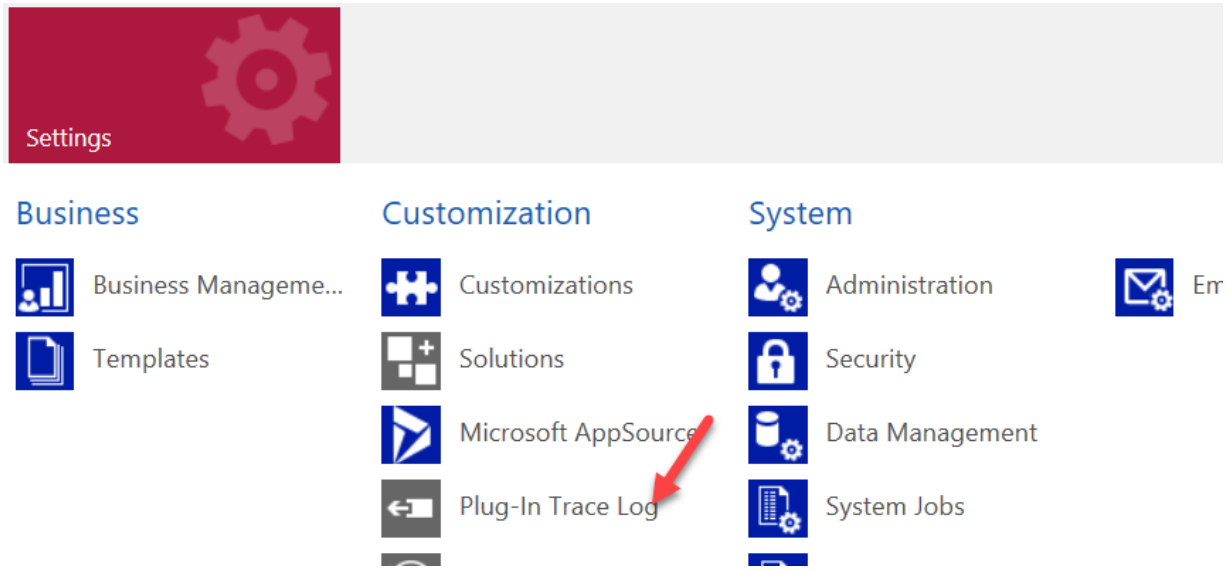
Objective: In this exercise, you will check the Plugin Trace log and debug the plugins

32.1 Task #1: Plugin Trace Log

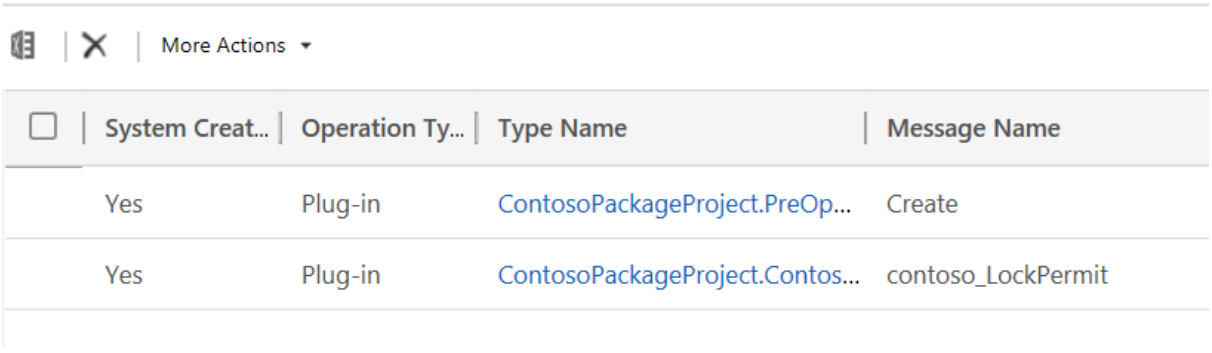
1. Open Plugin trace Log.
   - Go back to the Permit Management application.
   - Click **Settings** and select **Advanced Settings**.



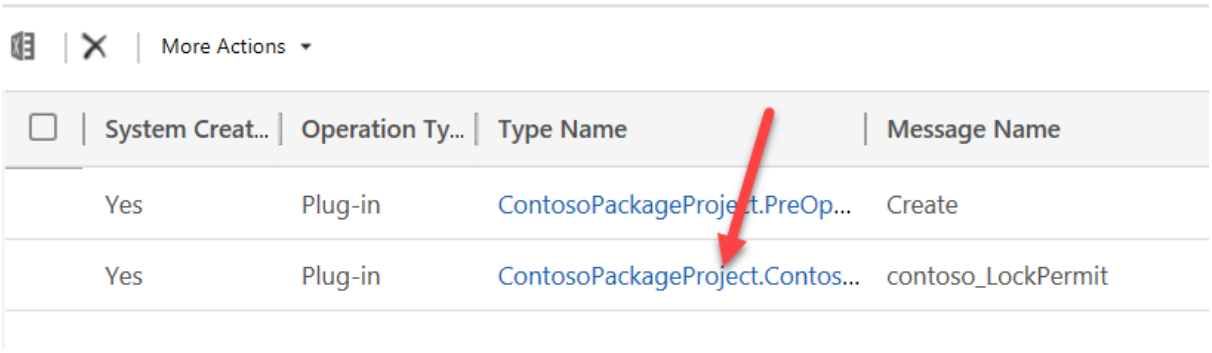
   - Click **Settings** and select **Plugin Trace Log**.



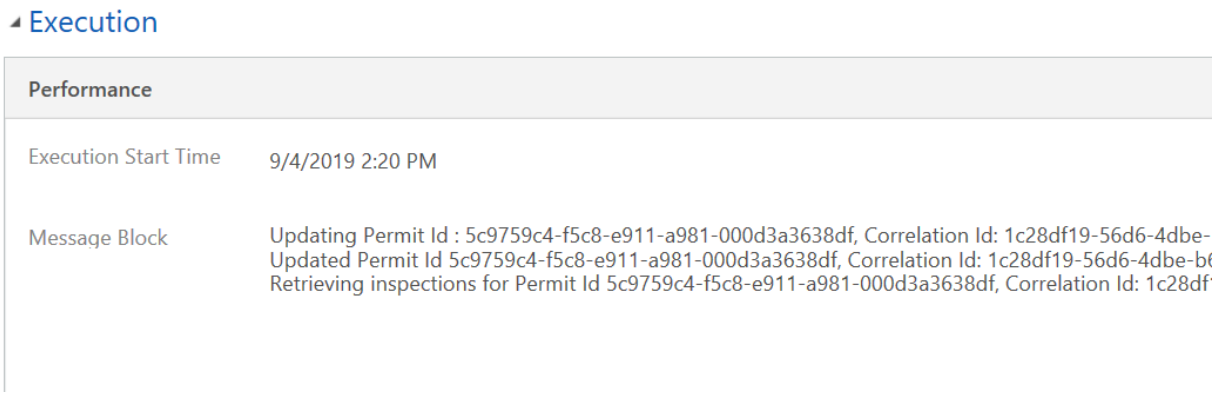
- You should see at least two logs.



2. Open the log and see what was logged.
   - Click to open the Lock Permit log.



   - Scroll down to the Execution section.
   - Examine your Trace messages.



32.2 Task #2: Debugging Plugins (Optional)

32.3 Follow these steps to debug your plugins https://docs.microsoft.com/en-us/powerapps/developer/common-data-service/tutorial-debug-plug-in

32.4 lab: title: 'Lab 07: Azure Functions'

[!NOTE] Effective November 2020:

- Common Data Service has been renamed to Microsoft Dataverse. Learn more
- Some terminology in Microsoft Dataverse has been updated. For example, *entity* is now *table* and *Column* is now *column*. Learn more

This content will be updated soon to reflect the latest terminology.

32.5 Lab 07 – Azure Functions

33 Scenario

A regional building department issues and tracks permits for new buildings and updates for remodeling of existing buildings. Throughout this course you will build applications and automation to enable the regional building department to manage the permitting process. This will be an end-to-end solution which will help you understand the overall process flow.

In this lab you will create an Azure Function that will handle routing inspections. Every morning people call in to request inspections on their permits. They must call before 9:30 AM and once that period ends all the inspections for the day must be assigned and sequenced for the inspectors. To accomplish this, you will build an Azure Function that runs on a schedule, queries pending inspections and assigns them to inspectors. Given the limited time to complete the lab, we've simplified the routing and sequencing decisions.

34 High-level lab steps

As part of building the Azure Function, you will complete the following:

- Configure an application user for the app along with a security role
- Build the function logic to route the requests

Deploy the Azure Function

34.1 Things to consider before you begin

- Could we have used Dynamics 365 Universal Resource Scheduling instead of custom code?
- Could we have used Power Automate instead of custom code?
- Remember to continue working in your DEVELOPMENT environment. We'll move everything to production soon.

35 Exercise #1: Configure an application user

Objective: In this exercise, you will configure an application user that will be used to connect the Azure Function back to the Microsoft Dataverse.

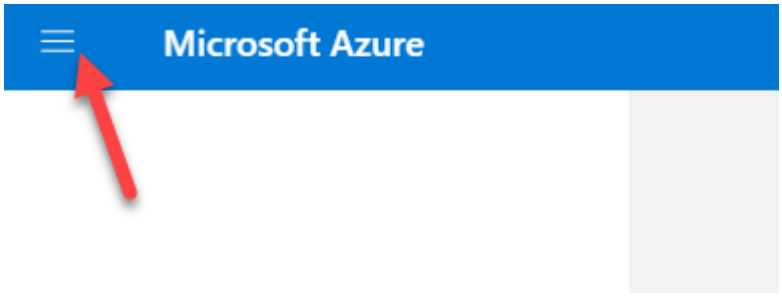
35.1 Task #1: Register Azure AD Application

1. Navigate to Azure Active Directory

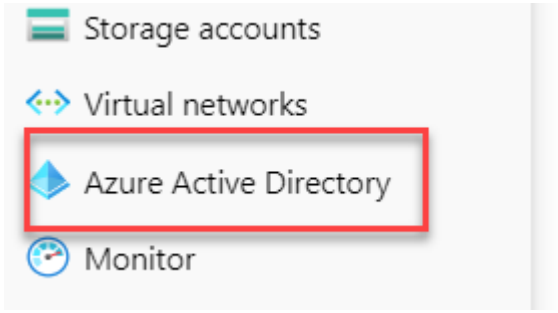
- Sign in to Azure Portal.

Note: You must be logged in with an organization account in the same tenant as your Microsoft Dataverse Environment. This does **NOT** have to be the account that has your Azure subscription.

- Click Show portal menu.



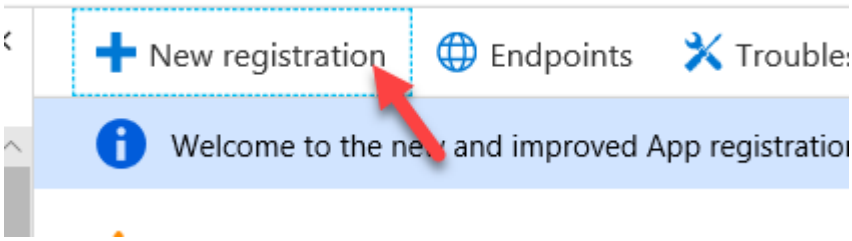
- Select **Azure Active Directory**.



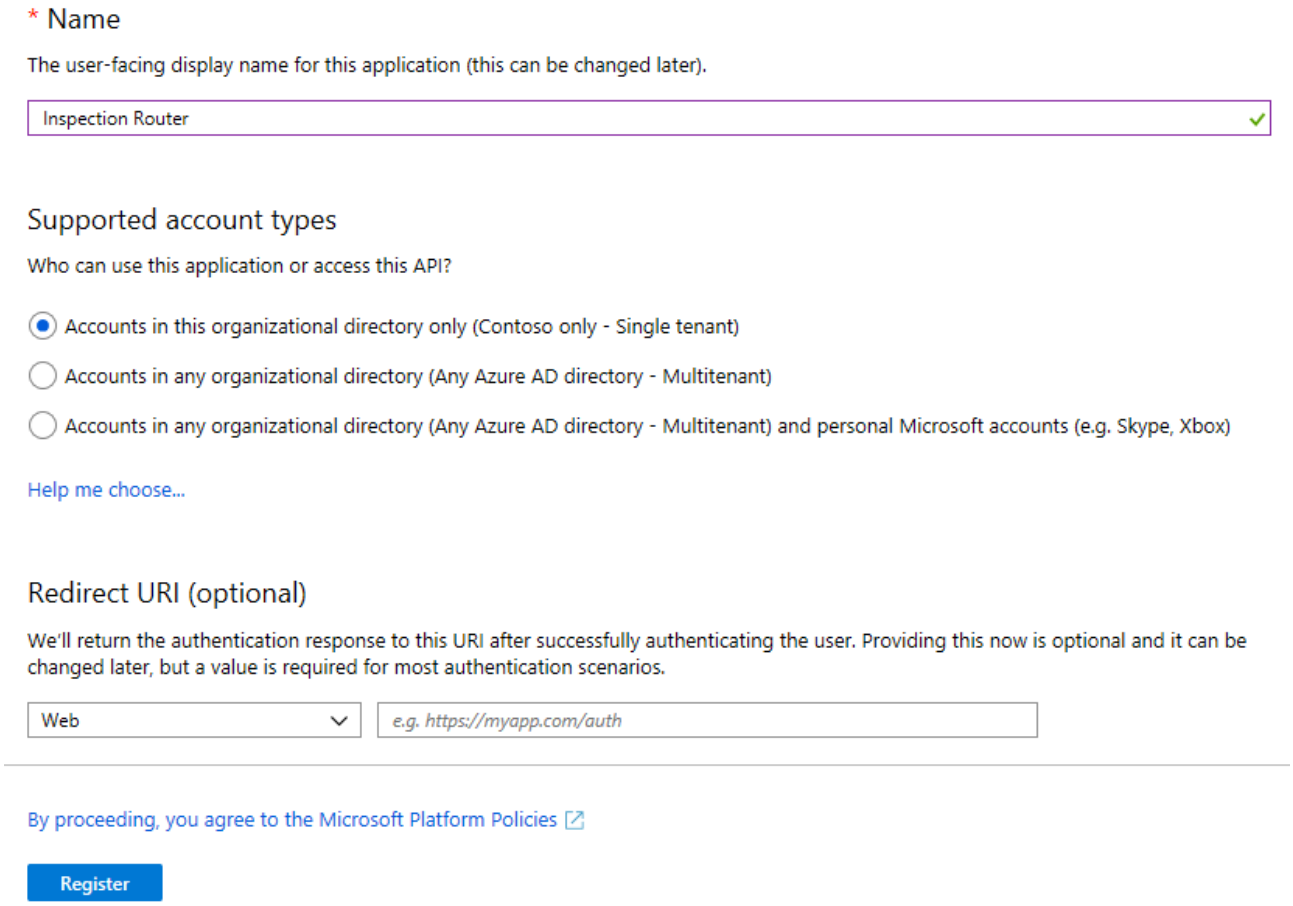
- Select **App Registration** from the **Manage** section.

2. Create new application registration.

- Click + **New registration**.



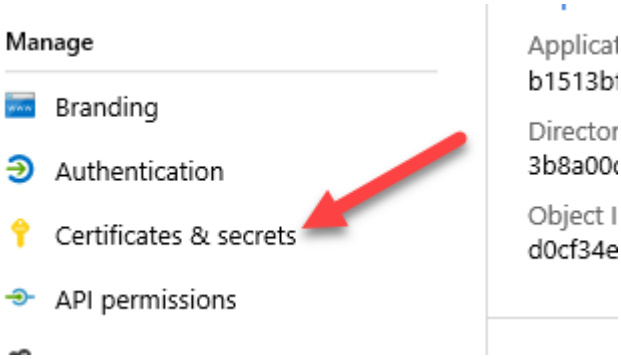
- Enter **Inspection Router** for **Name**, select **Accounts in this Organizational Directory Only** for **Supported Account Types**, and click **Register**.



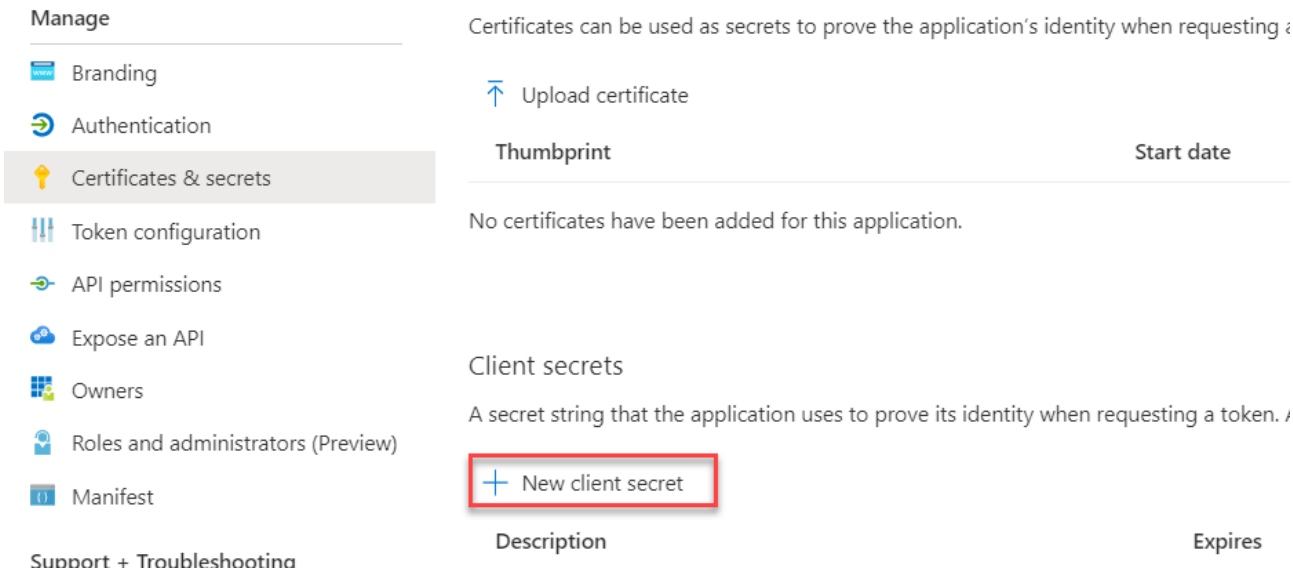
3. Add Client Secret.

Note: In this lab we are using a secret, however, you can also use a certificate. With either of these options you need to have a plan in place to handle rollover when they expire to ensure that the app keeps running.

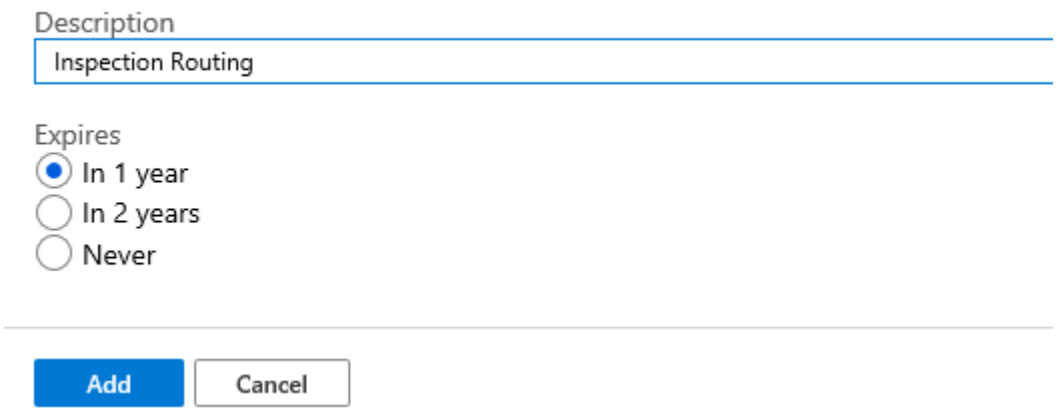
- Select **Certificates & Secrets**.



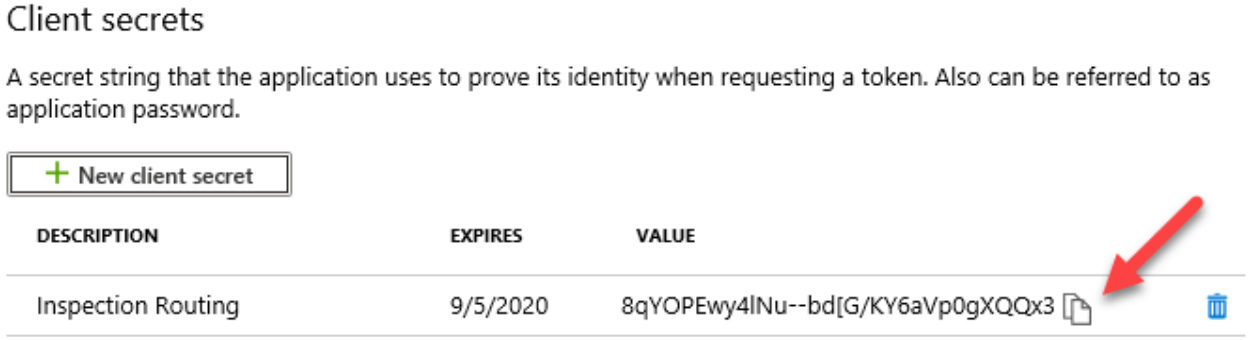
- Click + **New Client Secret**.



- Enter **Inspection Routing** for **Description** and click **Add**.



- Copy the **Value** and save it on a notepad. You need this value in future tasks.

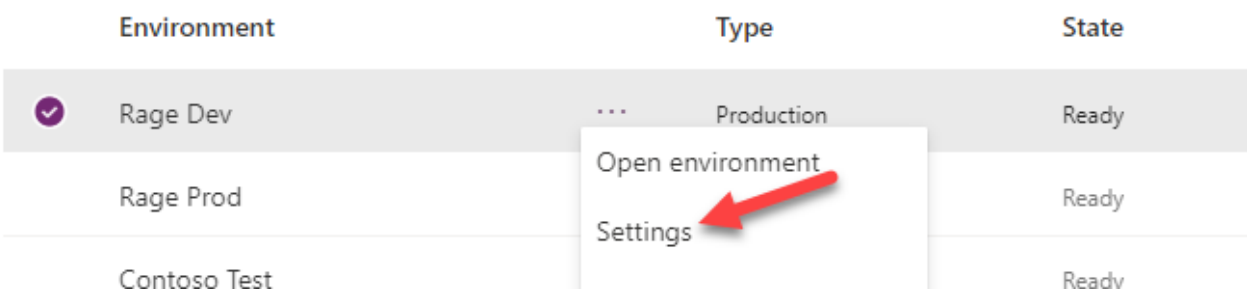


35.2 Task #2: Create Application User and Security Role

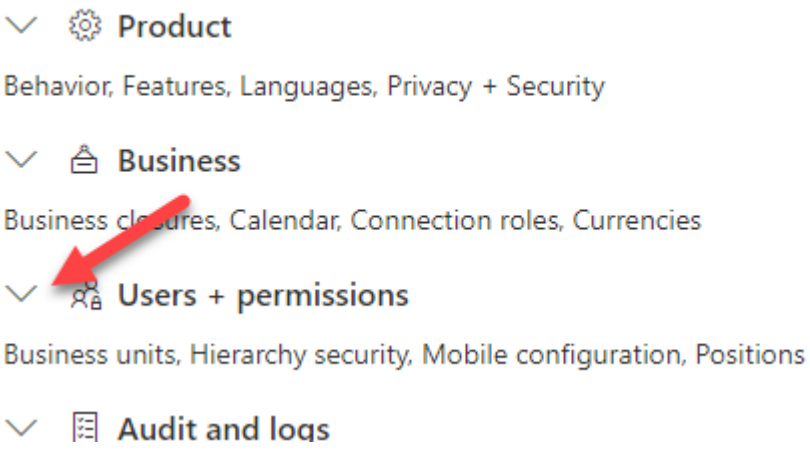
In this task, you will create the application user and associate it with the Azure AD app that you just registered. You will also create a security role needed to run the routing logic.

1. Navigate to security settings of your Dev environment

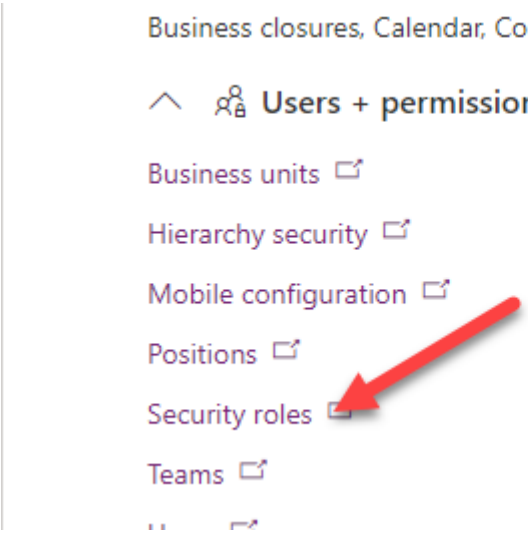
- Sign in to Power Platform admin center
- Select **Environments**.
- Select the **Dev** environment, click on the **... More Environment Actions** button and select **Settings**.



- Expand **Users + Permissions**.

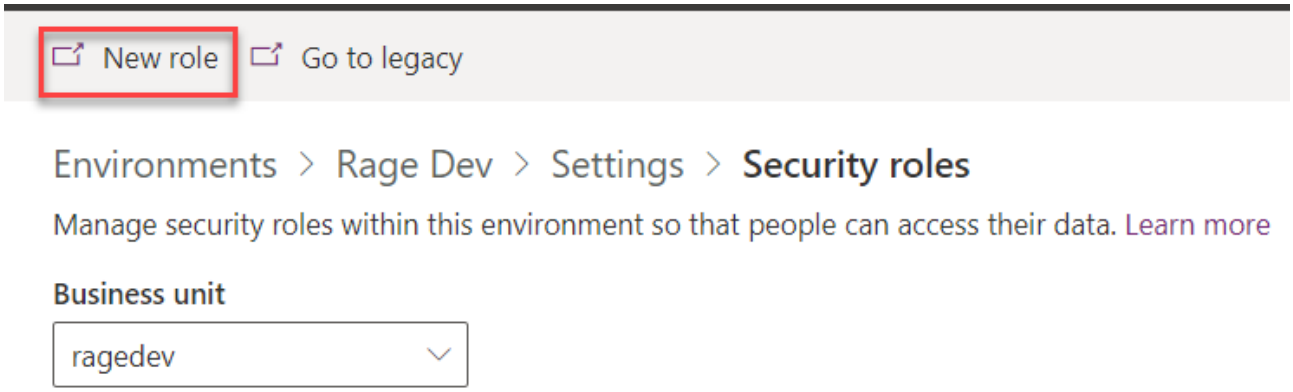


- Select **Security Roles**.

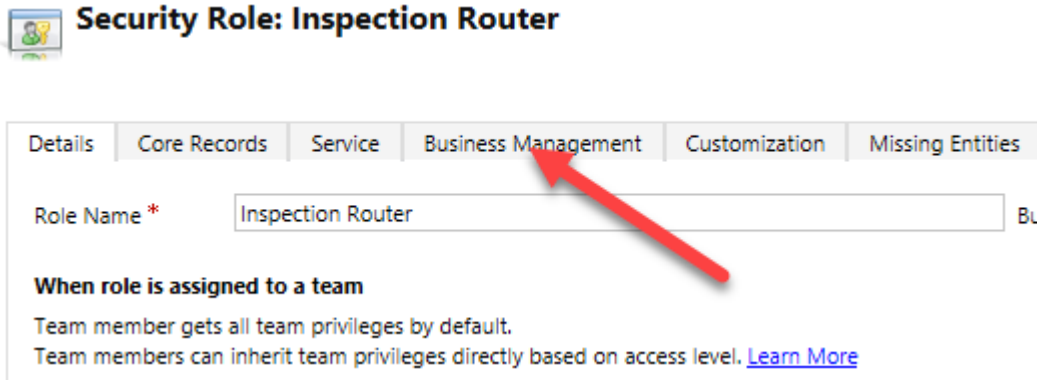


2. Create New Security Role

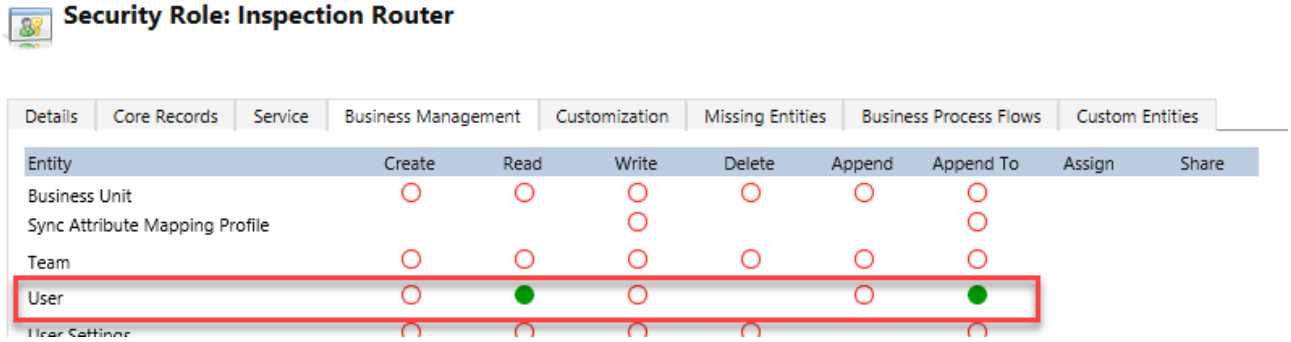
- Click **New role**.



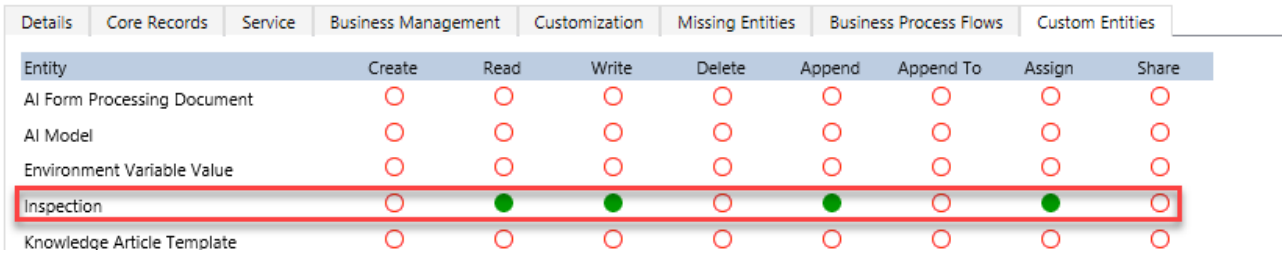
- Enter **Inspection Router** for **Role Name** and click **Save**.
- Select the **Business Management** tab.



- Locate the **User** Table and set **Read** and **Append To** privileges to **Organization**.



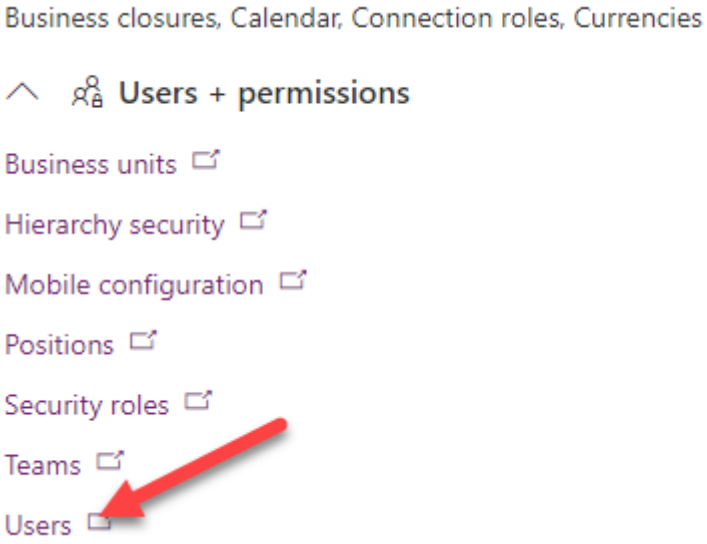
- Select the **Custom Tables** tab.
- Locate the **Inspection** Table and set **Read**, **Write**, **Append,** and **Assign** privileges to **Organization**.



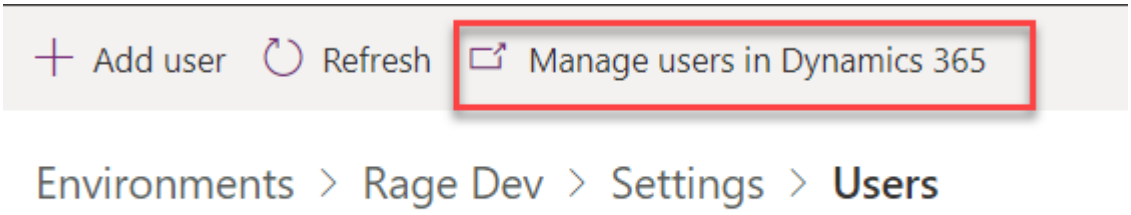
- Click **Save and Close**.
- Close the **Security Roles** browser tab/window.
- Click on the browser back button.

3. Navigate to the Application User form

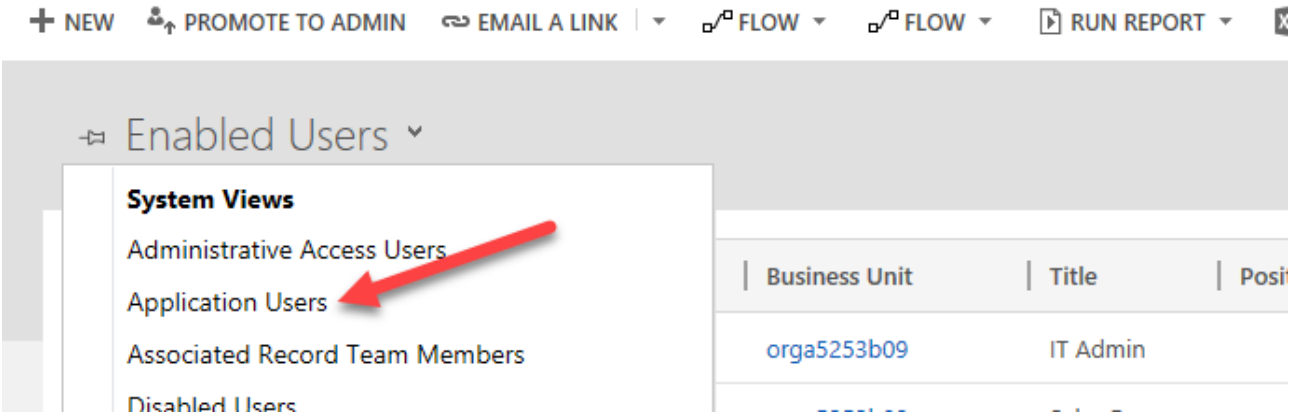
- Expand **Users + permissions** again.
- Click **Users**.



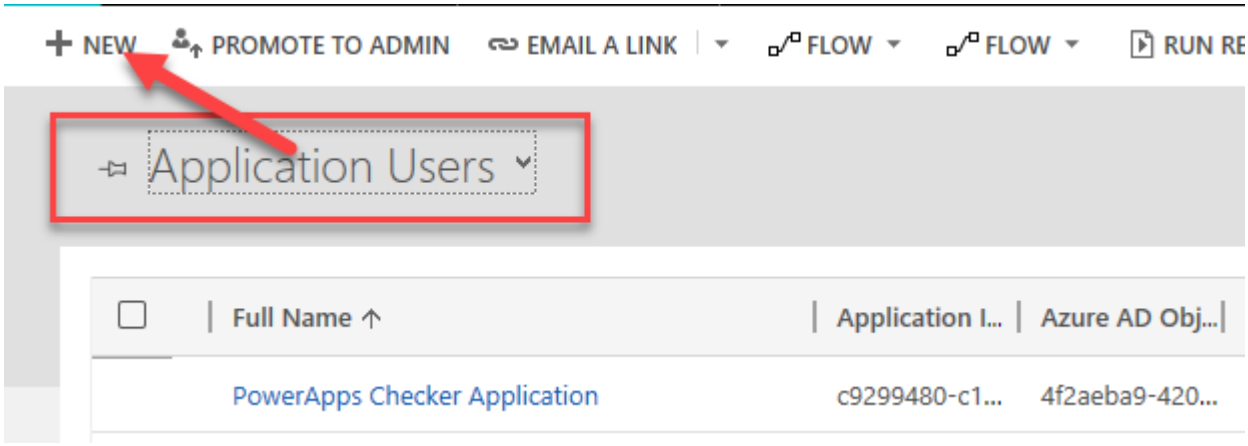
- Click **Manage users in Dynamics 365**.



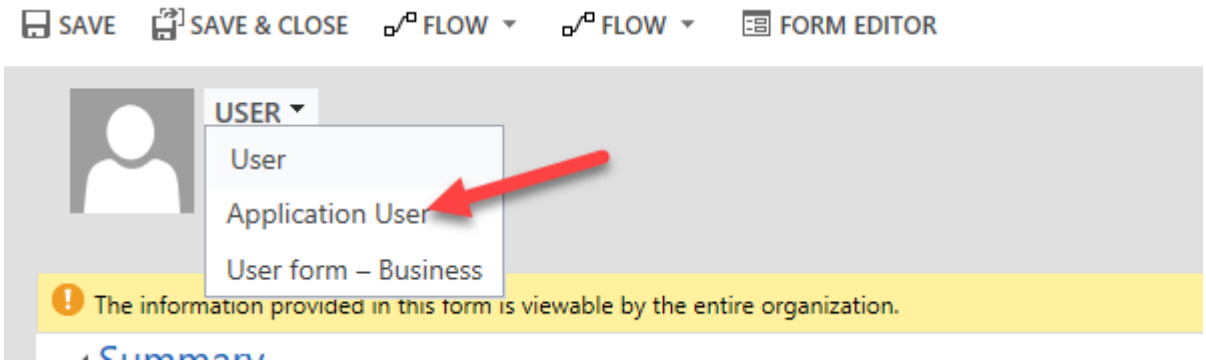
- Switch to the **Application Users** view.



- Click **+ New**.

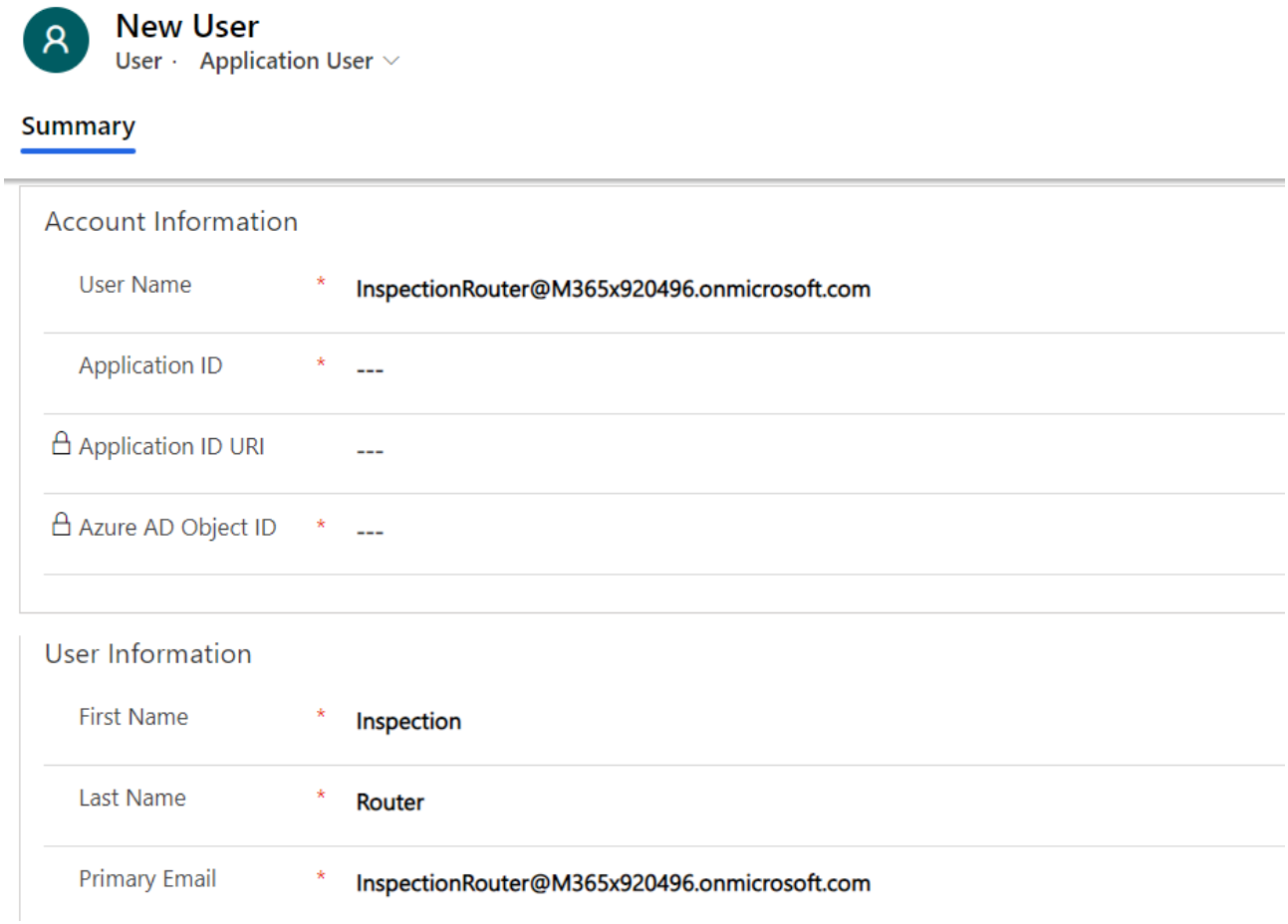


- Switch to the **Application User** form.



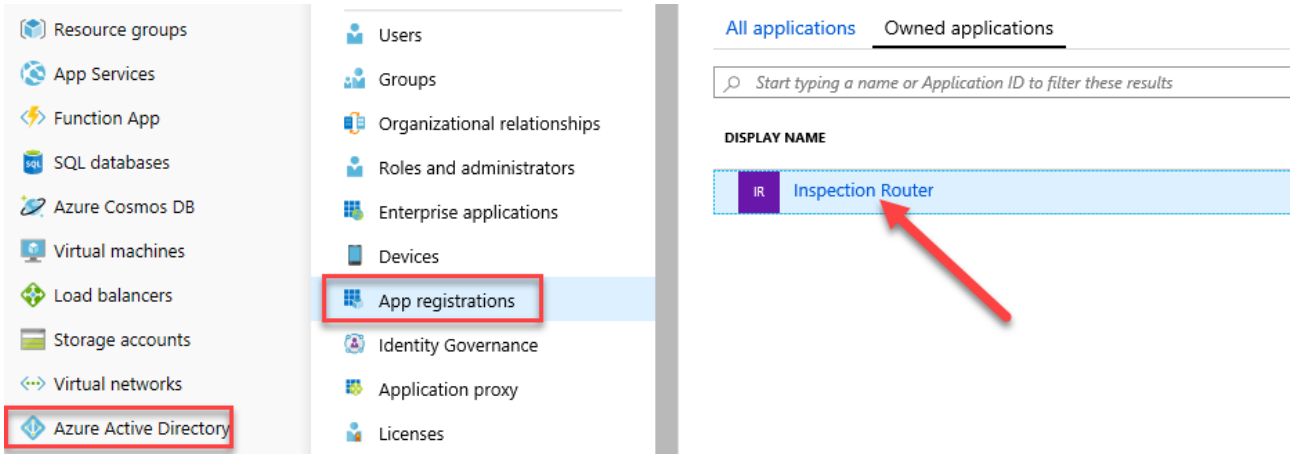
4. Create Application User.

- Click on the **Full Name** Column.
- Enter InspectionRouter@Tenant.onmicrosoft.com for User name, **Inspection** for **First Name**, **Router** for **Last Name**, enter InspectionRouter@Tenant.onmicrosoft.com for **Email**.

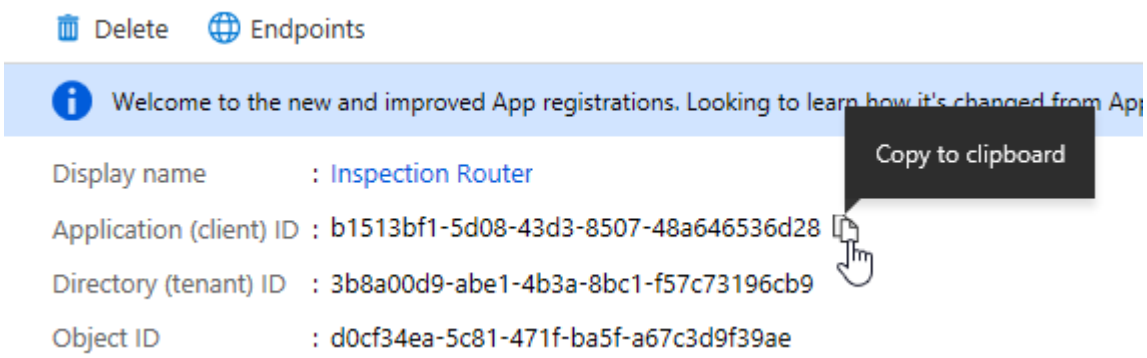


5. Copy App ID from Azure.

- Go back to your **Azure** portal.
- Select **Azure Active Directory**.
- Select **App Registrations | Owned applications**
- Click to open the registration you created.

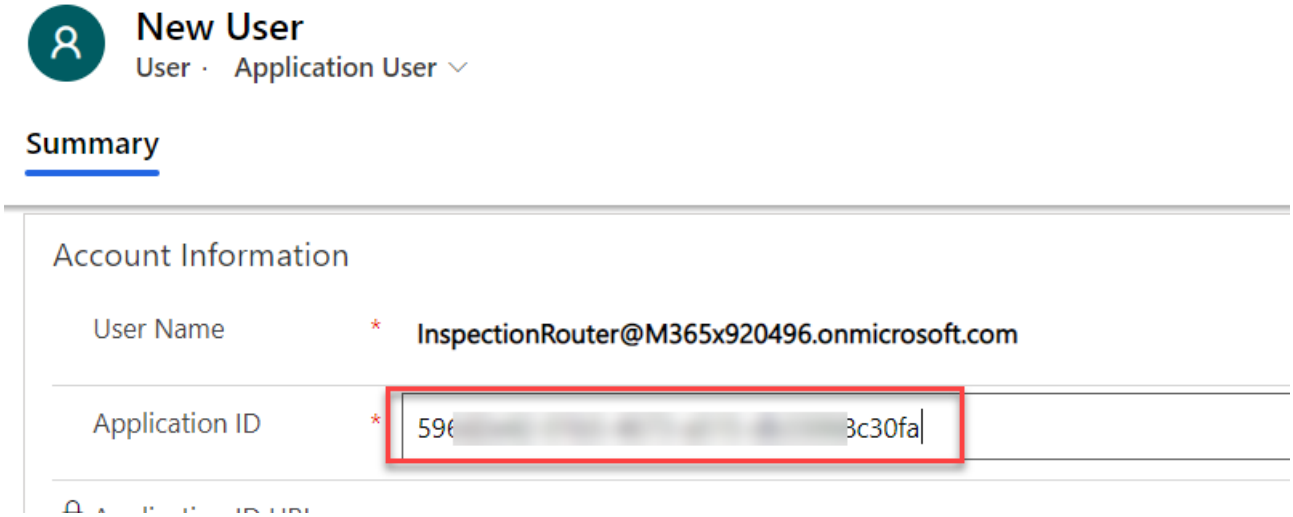


- Copy the **Application (Client ID).**

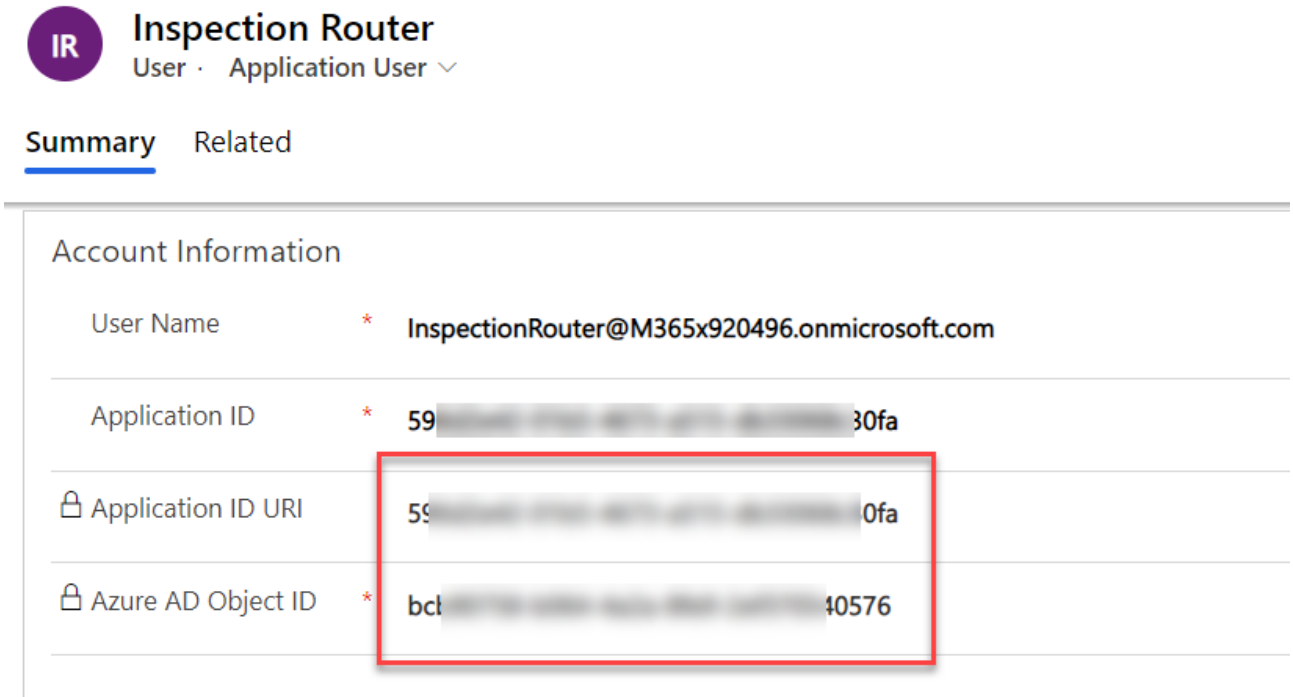


6. Complete the Application User creation

- Paste the Application Id you copied.



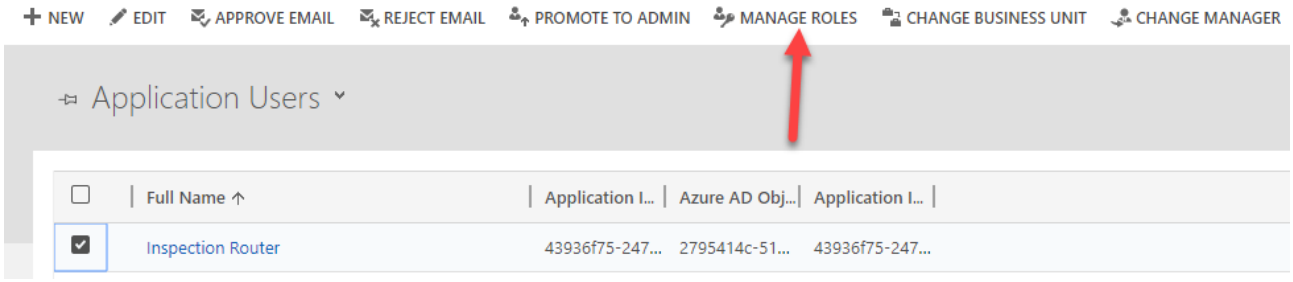
- Click **Save**.
- The **Application ID URI** and **Azure AD Object ID** Columns should auto populate.



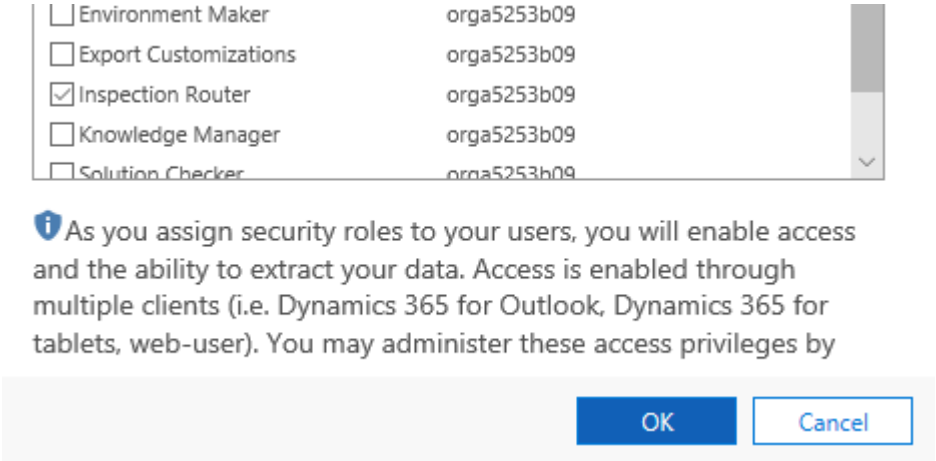
- Close the **User** form.

7. Assign the Inspection Router security role to the Application User you created

- Refresh the user view, select the user you create, and click **Manage Roles**.



- Select **Inspection Router** and click **OK**.



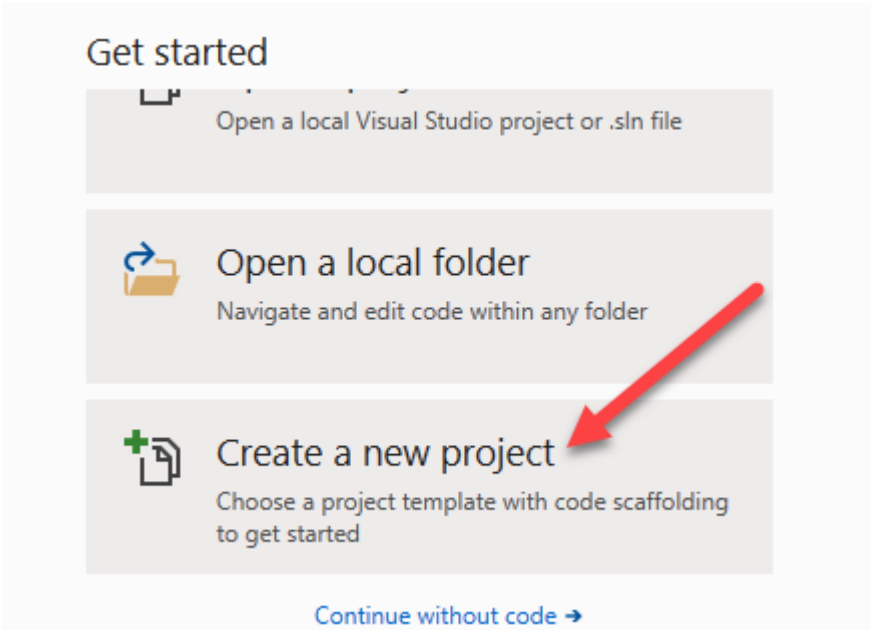
36 Exercise #2: Create Azure Function for Inspection Routing

Objective: In this exercise, you will create the Azure function that will route the inspections.

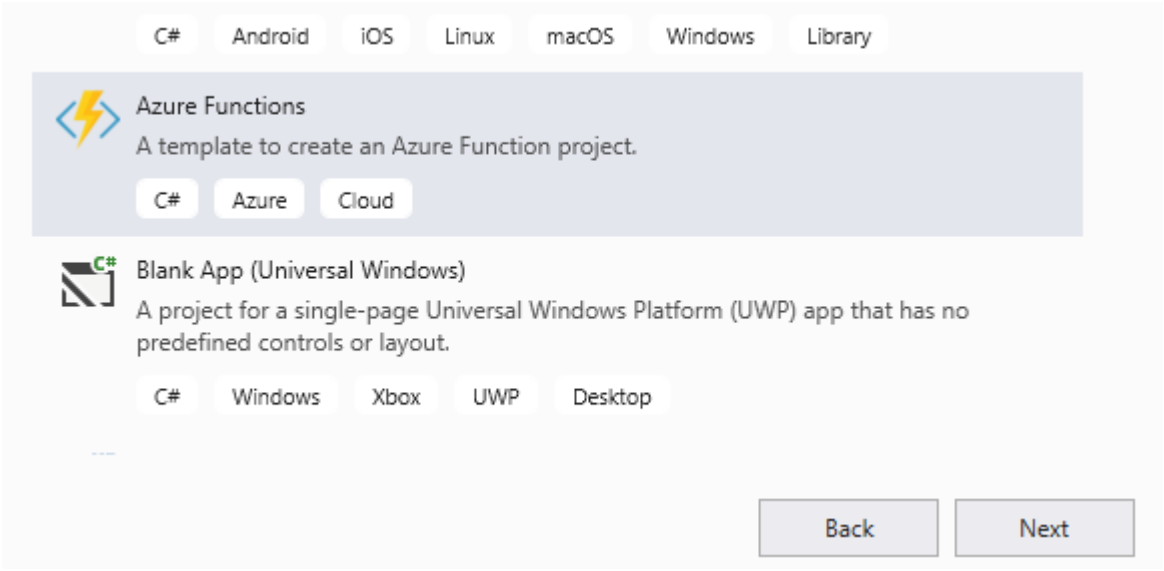
36.1 Task #1: Create the Function

1. Create Azure Function project

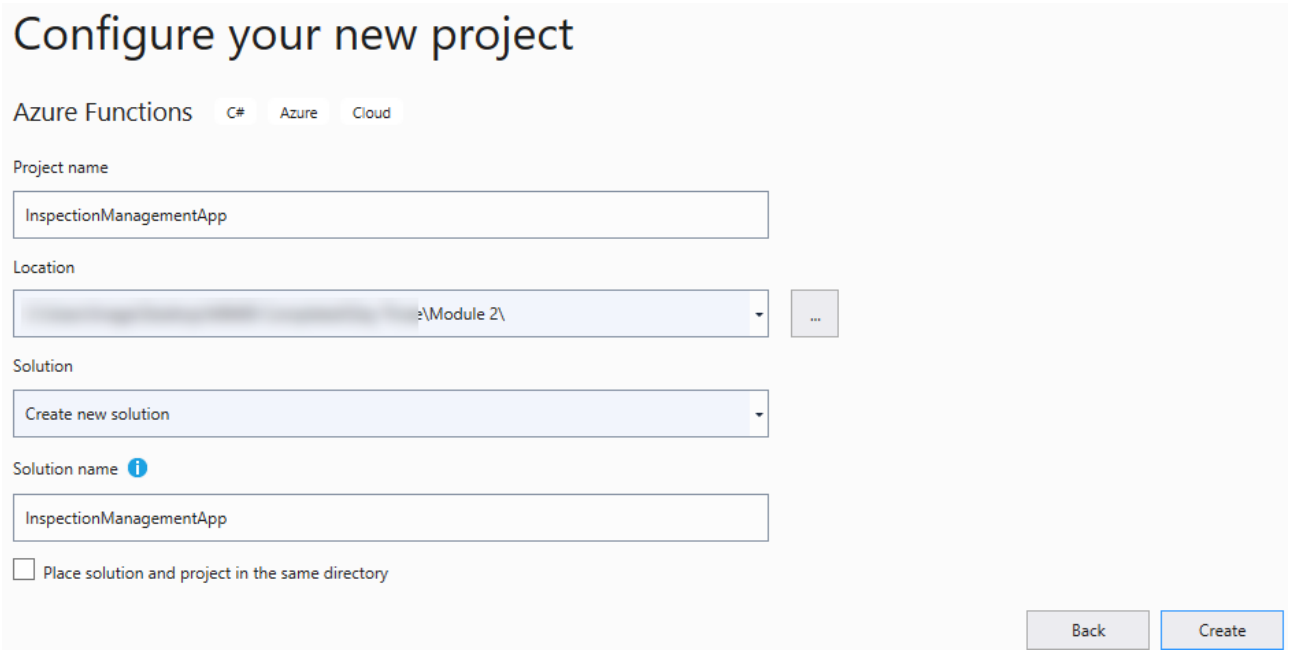
- Start **Visual Studio**.
- Click **Create a New Project**.



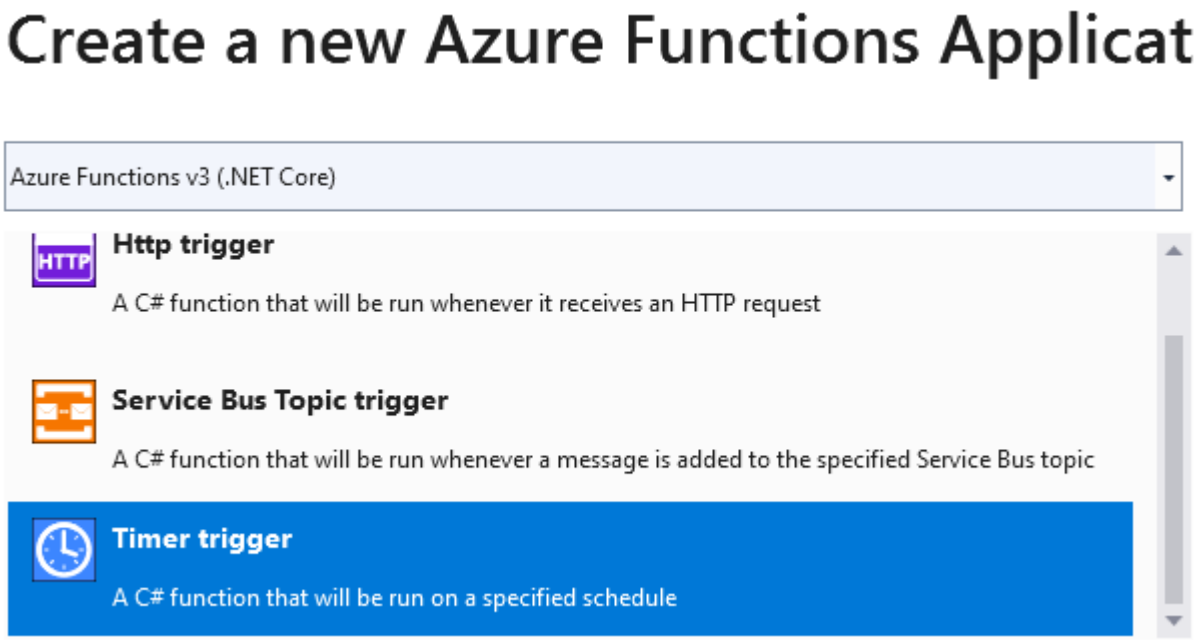
- Select **Azure Functions** and click **Next**.



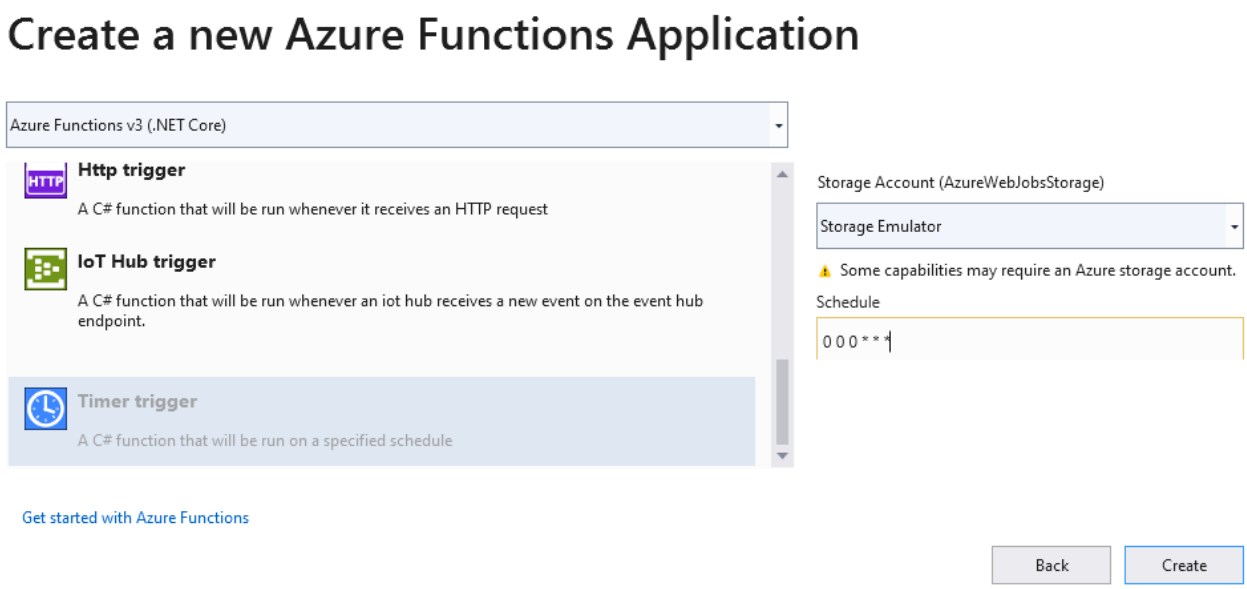
- Enter **InspectionManagementApp** for **Name** and click **Create**.



- Select **Azure Function V3 (.NET Core)** and select **Timer Trigger**.

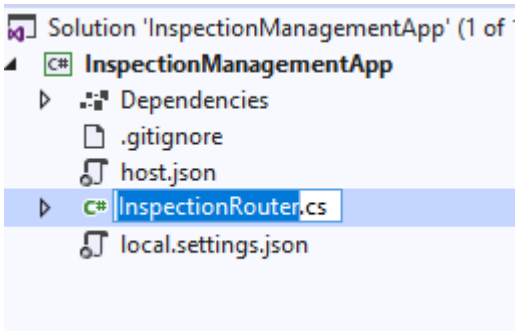


- Change the Schedule to **0 0 0** *** (Midnight Every Day) and click **Create**.

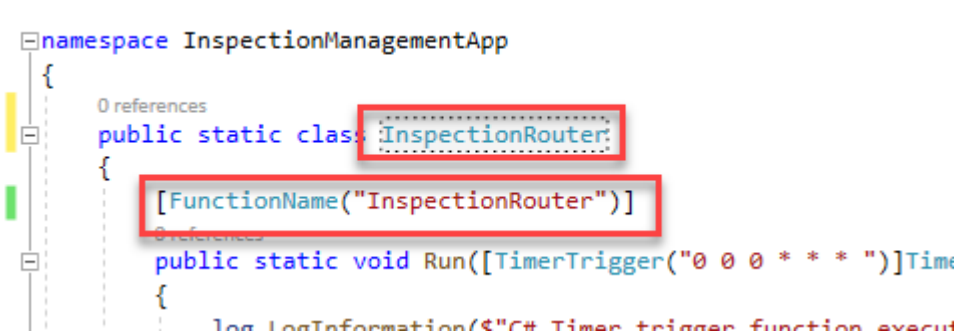


2. Rename and run the Function

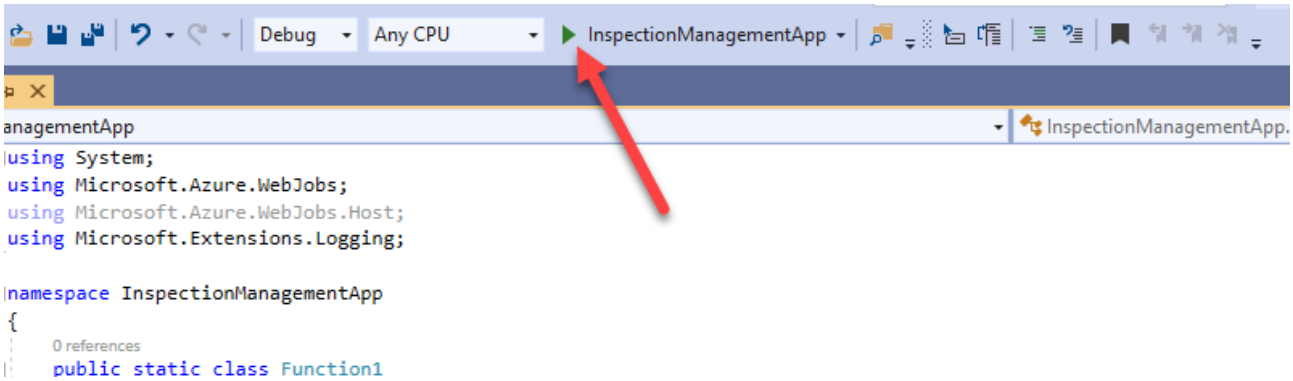
- Rename the function file **InspectionRouter**.



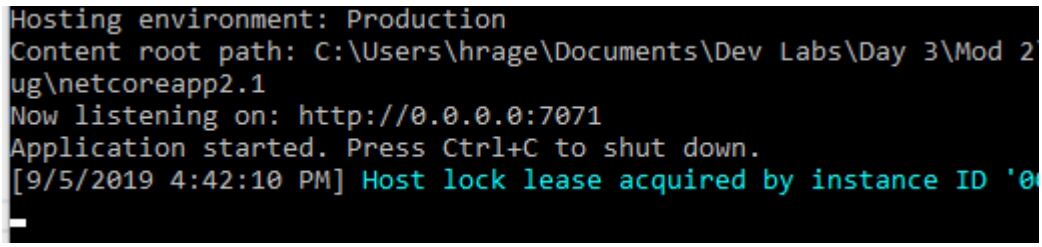
- Change the function class and FunctionName to **InspectionRouter.**



- Click **Run**.

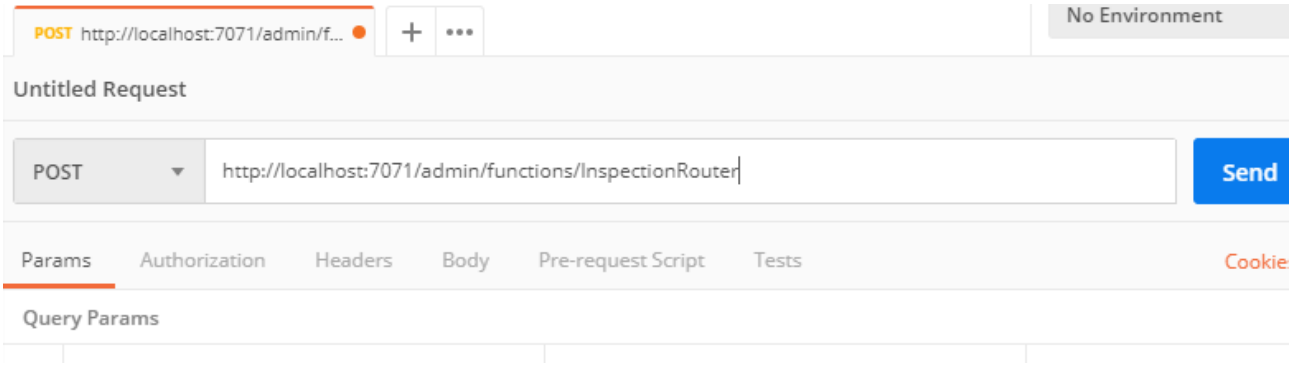


- You should see **Now Listening on:** http://0.0.0.0:7071

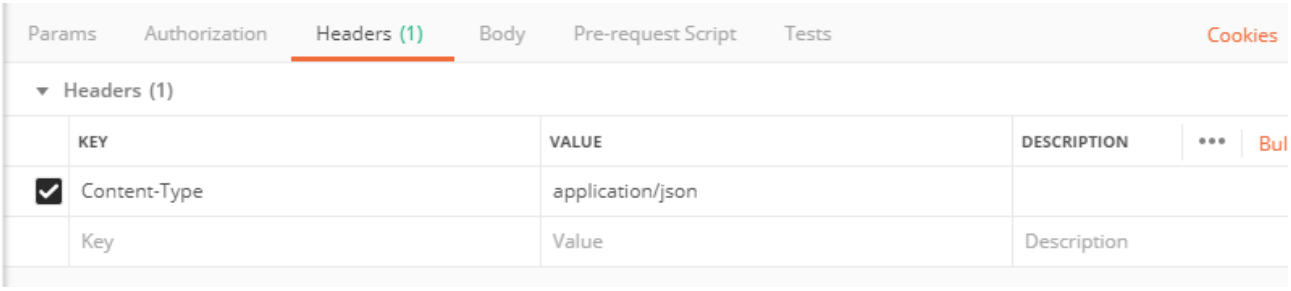


3. Trigger the function with Postman

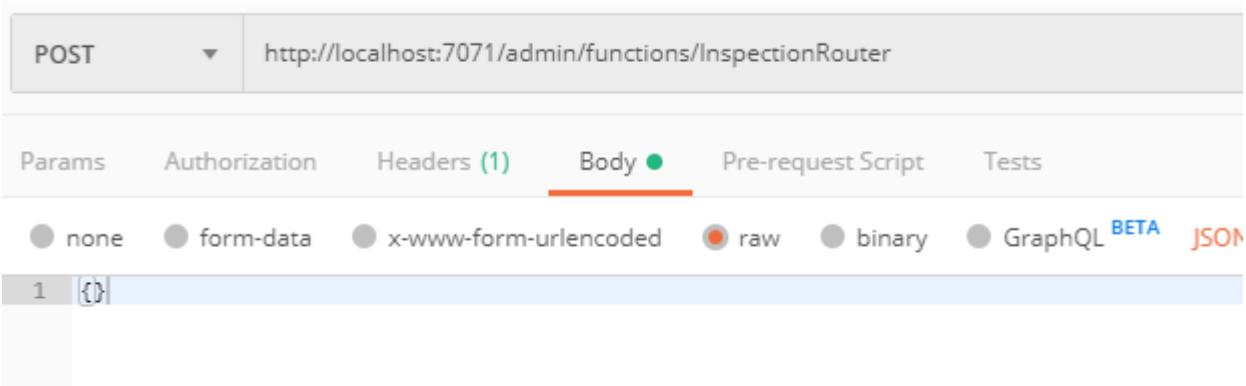
- If you don't have **Postman** installed, get it from here Postman and install it on your machine. Or you can use any similar tool of your preference that allows you to construct an HTTP POST request.
- Start **Postman**.
- Select **POST** and paste the URL: http://localhost:7071/admin/functions/InspectionRouter



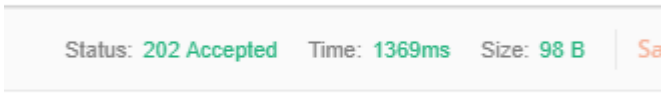
- Select the **Headers** tab.
- Add **Content-Type** and set it to **application/json**.



- Select the **Body** tab.
- Select **Raw** and set it to empty json **{}**.



- Click **Send**.
- You should get **202 Accepted Status**.



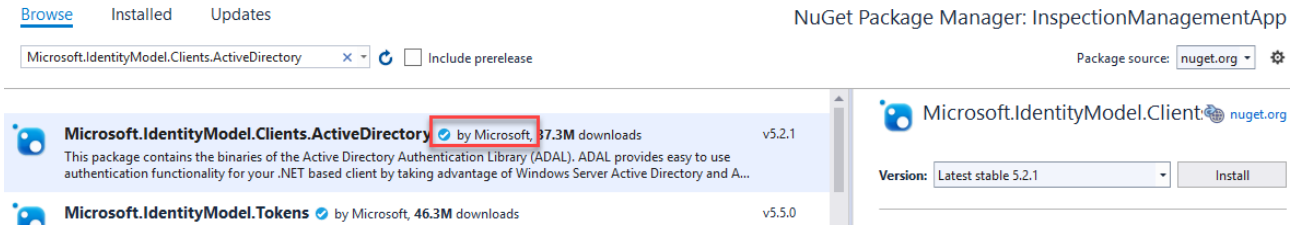
- Go to the output console.
- The function should get triggered.

```
9/5/2019 5:09:25 PM]        "level": "Admin"
9/5/2019 5:09:25 PM]      }
9/5/2019 5:09:25 PM]    ],
9/5/2019 5:09:25 PM]    "status": 202,
9/5/2019 5:09:25 PM]    "duration": 659
9/5/2019 5:09:25 PM] }
9/5/2019 5:09:25 PM] Executing 'InspectionRouter' (Reason='This function was programmatically called
, Id=848cc9cc-32d2-4e27-b3f3-3925bb7f3556)
9/5/2019 5:09:25 PM] C# Timer trigger function executed at: 9/5/2019 11:09:25 AM
9/5/2019 5:09:25 PM] Executed 'InspectionRouter' (Succeeded, Id=848cc9cc-32d2-4e27-b3f3-3925bb7f3556)
```

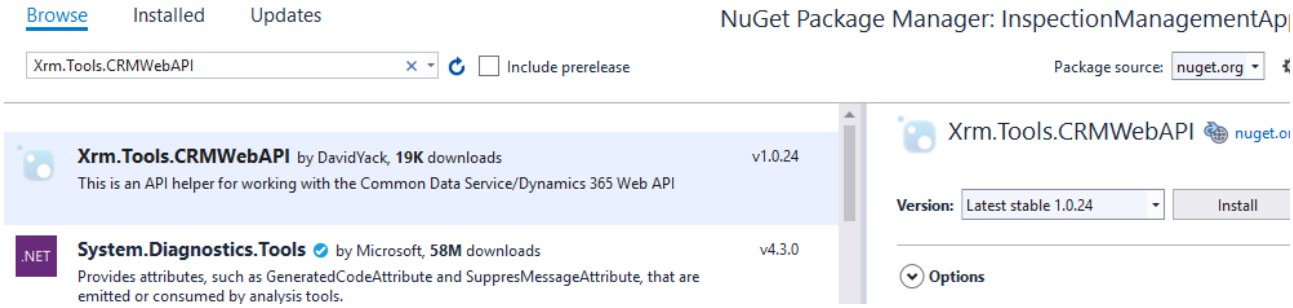
- Go back to **Visual Studio** and stop debugging.

4. Add NuGet packages

- Right Click on the project and select **Manage NuGet Packages**.
- Select the **Browse** tab and search for **Microsoft.IdentityModel.Clients.ActiveDirectory**.
- Select the latest stable version and click **Install**.



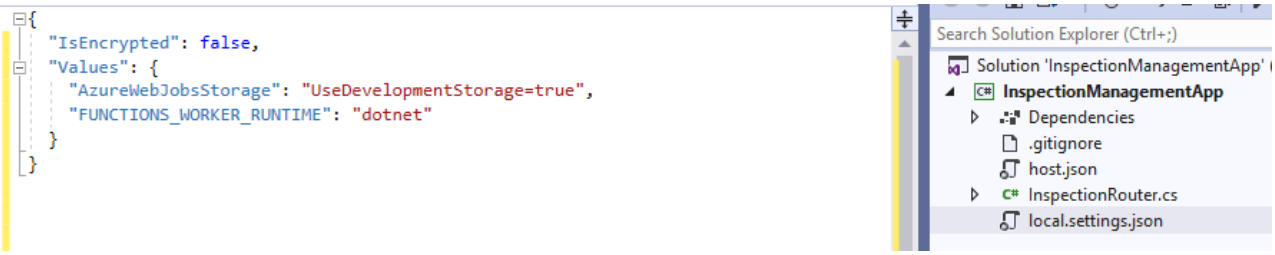
- Search for **Xrm.Tools.CrmWebAPI**. Note: This is a community library designed to work with the Microsoft Dataverse Web API. When you are building this type of extension you can use any oData V4 library you prefer. Make sure you select the one developed by DavidYack.
- Select the latest stable version and click **Install**.



- Close the **NuGet Package Manager**.

5. Edit the local settings file

- Click to open the **local.settings.json** file



- Add the **Values** below to **local.settings**

```
"cdsurl": "",

"cdsclientid": "",

"cdsclientsecret": ""
```

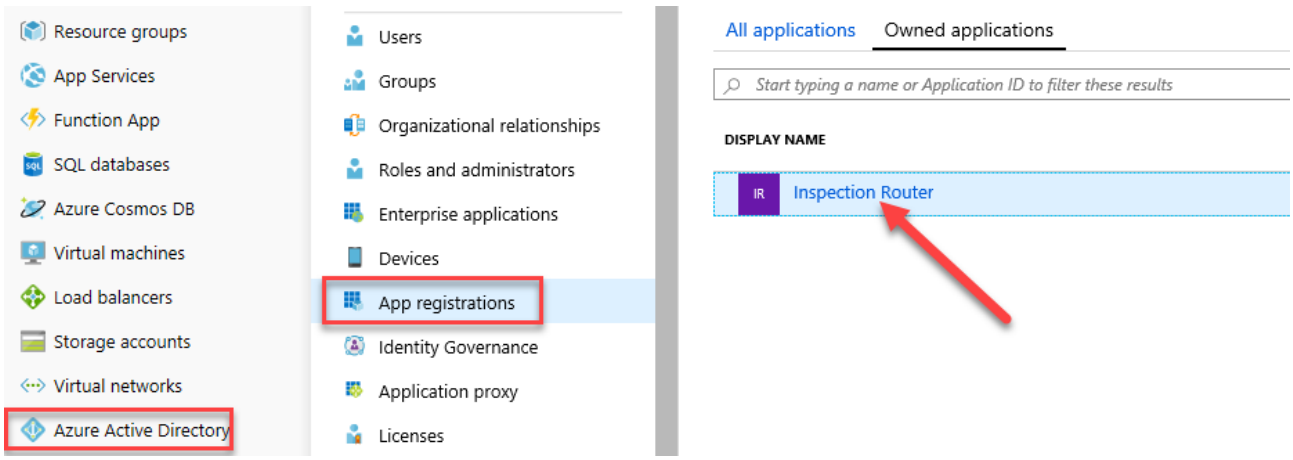
```
{
"IsEncrypted": false,
"Values": {
"AzureWebJobsStorage": "UseDevelopmentStorage=true",
"FUNCTIONS_WORKER_RUNTIME": "dotnet",
"cdsurl":  "",
"cdsclientid": "",
"cdsclientsecret":  ""
}
}
```

- Find the Client Secret you saved in the notepad and paste as the cdsclientsecret.

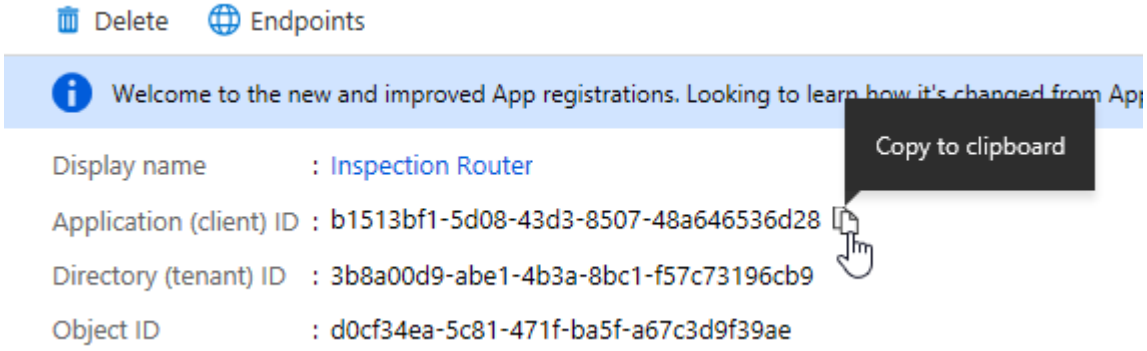
```
{
"IsEncrypted": false,
"Values": {
"AzureWebJobsStorage": "UseDevelopmentStorage=true",
"FUNCTIONS_WORKER_RUNTIME": "dotnet",
"cdsurl": "",
"cdsclientid": "",
"cdsclientsecret": "8qYOPEwy4lNu--bd[G/KY6aVp0gXQQx3"
}
}
```

6. Copy the App ID

- Go back to your **Azure** portal.
- Select **Azure Active Directory**.
- Select **App Registrations**.
- Click to open the registration you created.



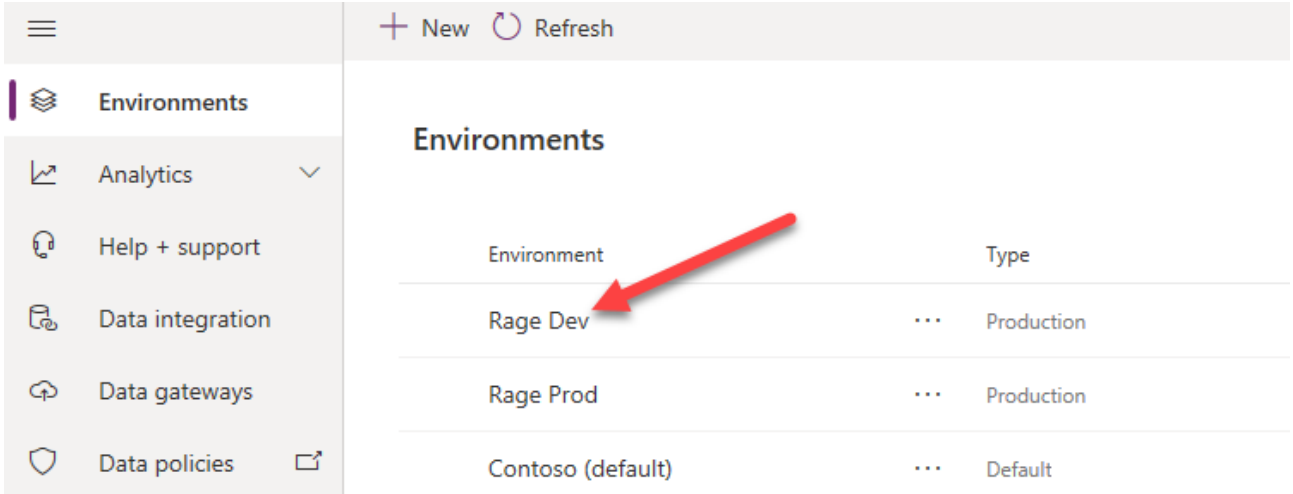
- Copy the **Application (Client ID).**



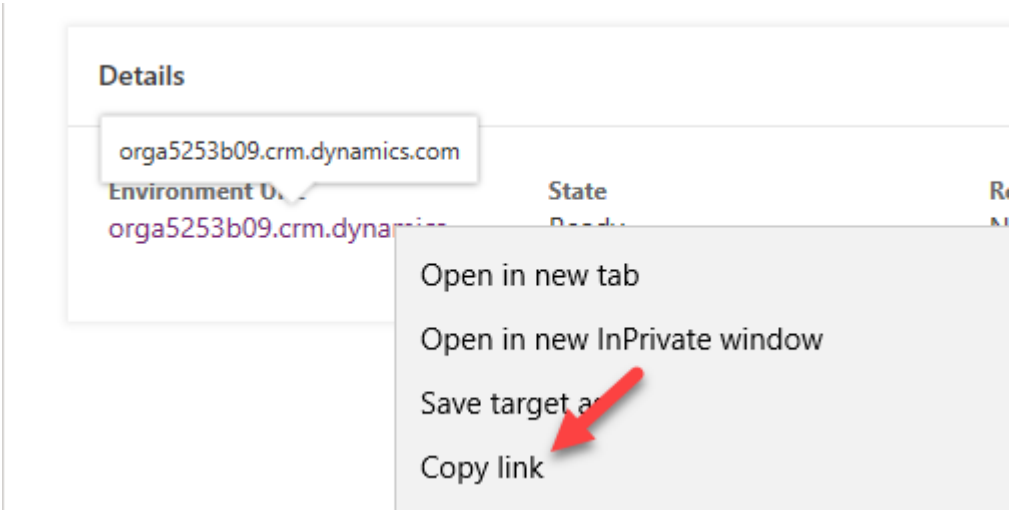
- Go back to **Visual Studio** and paste the **Application ID** as the **cdsclientid**.

7. Find the your Microsoft Dataverse URL

- Sign in to https://admin.powerplatform.microsoft.com
- Select **Environments** and click to open the **Dev** environment.



- Copy the **Environment URL**.



- Go back to **Visual Studio** and paste the **URL** you copied as the **cdsurl**.

```
{
  "IsEncrypted": false,
  "Values": {
    "AzureWebJobsStorage": "UseDevelopmentStorage=true",
    "FUNCTIONS_WORKER_RUNTIME": "dotnet",
    "cdsurl": "https://orga5253b09.crm.dynamics.com",
    "cdsclientid": "b1513bf1-5d08-43d3-8507-48a646536d28",
    "cdsclientsecret": "8qYOPEwy4lNu--bd[G/KY6aVp0gXQQx3"
  }
}
```

- Save and close the file.

8. Add using statements to the function class.

- Open the **InspectionRouter.cs** file
- Add the using statements below.

```
using System.Threading.Tasks;
using Xrm.Tools.WebAPI;
using Microsoft.IdentityModel.Clients.ActiveDirectory;
using Xrm.Tools.WebAPI.Results;
using System.Dynamic;
using Xrm.Tools.WebAPI.Requests;
using System.Collections.Generic;
```

9. Create a method that will create the web API.

- Add the method below inside the class.

```
private static async Task<CRMWebAPI> GetCRMWebAPI(ILogger log)
{
return null;
}
```

- Add the local variables below before the return line on the **GetCRMWebAPI** method.

```
var clientID = Environment.GetEnvironmentVariable("cdsclientid", EnvironmentVariableTarget.Proces
var clientSecret = Environment.GetEnvironmentVariable("cdsclientsecret", EnvironmentVariableTarg
var crmBaseUrl = Environment.GetEnvironmentVariable("cdsurl", EnvironmentVariableTarget.Process)
var crmurl = crmBaseUrl + "/api/data/v9.0/";
```

- Create **Authentication Parameters**.

```
AuthenticationParameters ap = await AuthenticationParameters.CreateFromUrlAsync(new Uri(crmurl))
```

- Create **Client Credential** passing your **Client Id** and **Client Secret**.

```
var clientcred = new ClientCredential(clientID, clientSecret);
```

- Get **Authentication Context**.

```
// CreateFromUrlAsync returns endpoint while AuthenticationContext expects authority
// workaround is to downgrade adal to v3.19 or to strip the tail
var auth = ap.Authority.Replace("/oauth2/authorize", "");
var authContext = new AuthenticationContext(auth);
```

- Get **Token**.

```
var authenticationResult = await authContext.AcquireTokenAsync(crmBaseUrl, clientcred);
```

- Return the **web API**. Replace the return line with the code below.

```
return new CRMWebAPI(crmurl, authenticationResult.AccessToken);
```

```
1 reference
private static async Task<CRMWebAPI> GetCRMWebAPI(ILogger log)
{
    var clientID = Environment.GetEnvironmentVariable("cdsclientid", EnvironmentVariableTarget.Process);
    var clientSecret = Environment.GetEnvironmentVariable("cdsclientsecret", EnvironmentVariableTarget.Process);
    var crmBaseUrl = Environment.GetEnvironmentVariable("cdsurl", EnvironmentVariableTarget.Process);
    var crmurl = crmBaseUrl + "/api/data/v9.0/";

    AuthenticationParameters ap = await AuthenticationParameters.CreateFromUrlAsync(new Uri(crmurl));

    var clientcred = new ClientCredential(clientID, clientSecret);

    // CreateFromUrlAsync returns endpoint while AuthenticationContext expects authority
    // workaround is to downgrade adal to v3.19 or to strip the tail
    var auth = ap.Authority.Replace("/oauth2/authorize", "");
    var authContext = new AuthenticationContext(auth);

    var authenticationResult = await authContext.AcquireTokenAsync(crmBaseUrl, clientcred);

    return new CRMWebAPI(crmurl, authenticationResult.AccessToken);
}
```

10. Test the web API you created

- Call the GetCRMWebAPI method. Add the code below to the Run method.

```
CRMWebAPI api = GetCRMWebAPI(log).Result;
```

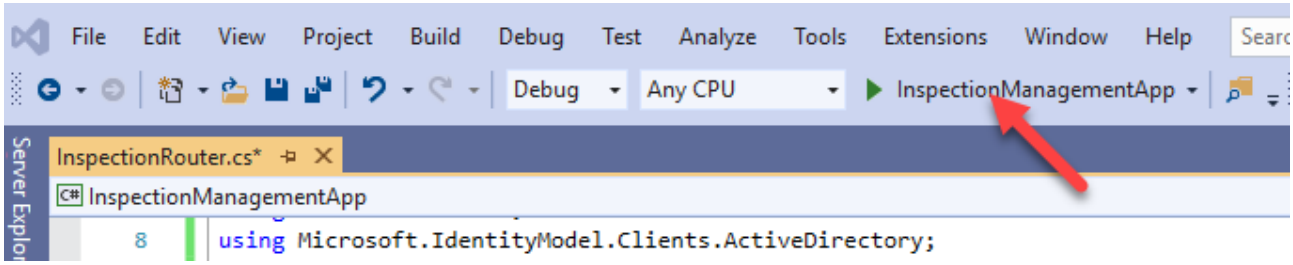
- Execute **WhoAmI** function and log the **User Id**.

```
dynamic whoami = api.ExecuteFunction("WhoAmI").Result;
log.LogInformation($"UserID: {whoami.UserId}");
```

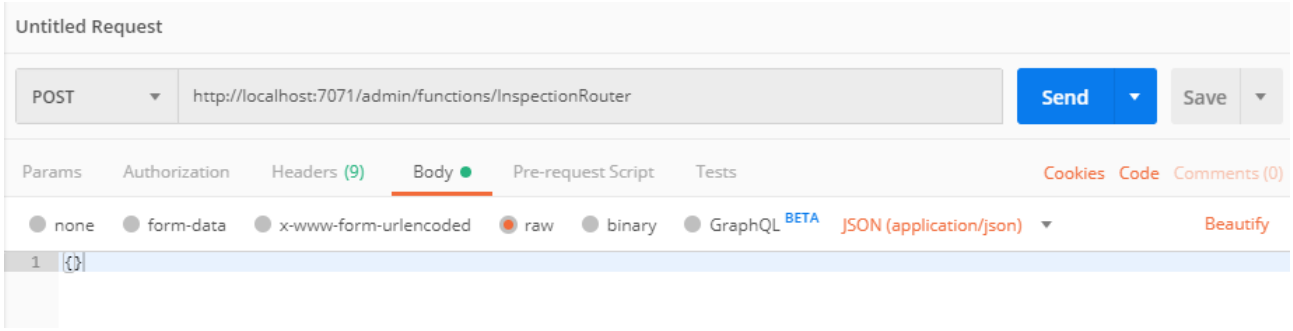
```
[FunctionName("InspectionRouter")]
0 references
public static void Run([TimerTrigger("0 0 0 * * * ")]TimerInfo myTimer, ILogger log)
{
    log.LogInformation($"C# Timer trigger function executed at: {DateTime.Now}");

    CRMWebAPI api = GetCRMWebAPI(log).Result;
    dynamic whoami = api.ExecuteFunction("WhoAmI").Result;
    log.LogInformation($"UserID: {whoami.UserId}");
}
```

11. Debug

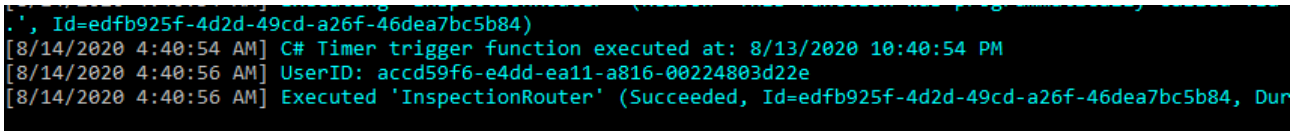
- Click Run.



- Go back to **Postman** and click **Send**.



- Go to the output console.
- You should see the **User ID**.



Go back **Visual Studio** and stop debugging.

36.2 Task #2: Get Inspections and Users and Assign Inspections

1. Create a method that will get all active inspections that are New Request or Pending, and scheduled for today

- Add the method below inside the class.

```
private static Task<CRMGetListResult<ExpandoObject>> GetInspections(CRMWebAPI api)

{

return null;

}
```

- Create **Fetch XML**. Add the code below before the return line of the GetInspections method.

```
var fetchxml = @"<fetch version=""1.0"" mapping=""logical"" >
<entity name=""contoso_inspection"" >
<attribute name=""contoso_inspectionid"" />
<attribute name=""contoso_name"" />
<attribute name=""ownerid"" />
<attribute name=""contoso_inspectiontype"" />
<attribute name=""contoso_sequence"" />
<attribute name=""contoso_scheduleddate"" />
<filter type=""and"" >
<condition value=""0"" operator=""eq"" attribute=""statecode"" />
<condition attribute=""contoso_scheduleddate"" operator=""today"" />
<condition attribute=""statuscode"" operator=""in"" >
<value>1</value>
<value>463270000</value>
</condition>
```

```
</filter>
</entity>
</fetch>";
```

- Get the list of Inspections.

```
var inspections = api.GetList<ExpandoObject>("contoso_inspections", QueryOptions: new CRMGetList
{
FetchXml = fetchxml
});
```

- Return the Inspections. Replace the return line with the code below.

```
return inspections;
```

```
private static async Task<CRMWebAPI> GetCRMWebAPI(ILogger log)...

0 references
private static Task<CRMGetListResult<ExpandoObject>> GetInspections(CRMWebAPI api)
{
    var fetchxml = @"<fetch version=""1.0"" mapping=""logical"" >
     <entity name=""contoso_inspection"" >
       <attribute name=""contoso_inspectionid"" />
       <attribute name=""contoso_name"" />
       <attribute name=""ownerid"" />
       <attribute name=""contoso_inspectiontype"" />
       <attribute name=""contoso_sequence"" />
       <attribute name=""contoso_scheduleddate"" />
       <filter type=""and"" >
         <condition value=""0"" operator=""eq"" attribute=""statecode"" />
         <condition attribute=""contoso_scheduleddate"" operator=""today"" />
         <condition attribute=""statuscode"" operator=""in"" >
           <value>1</value>
           <value>463270000</value>
         </condition>
       </filter>
     </entity>
    </fetch>";

    var inspections = api.GetList<ExpandoObject>("contoso_inspections", QueryOptions: new CRMGetListOptions()
    {
        FetchXml = fetchxml
    });

    return inspections;
}
```

2. Call the GetInspections method from the Run method.

- Go back to the **Run** method.
- Call the **GetInspections** method.

```
var inspections = GetInspections(api).Result;
```

```
[FunctionName("InspectionRouter")]
0 references
public static void Run([TimerTrigger("0 0 0 * * * ")]TimerInfo myTimer, ILogger log)
{
    log.LogInformation($"C# Timer trigger function executed at: {DateTime.Now}");

    CRMWebAPI api = GetCRMWebAPI(log).Result;
    dynamic whoami = api.ExecuteFunction("WhoAmI").Result;
    log.LogInformation($"UserID: {whoami.UserId}");

    var inspections = GetInspections(api).Result;
}
```

3. Create a method that will get all users.

- Add the method below inside the class.

```
private static Task<CRMGetListResult<ExpandoObject>> GetUsers(CRMWebAPI api)
{
var users = api.GetList<ExpandoObject>("systemusers");
return users;
}
```

- Call the **GetUsers** method from the **Run** method.

```
var users = GetUsers(api).Result;
```

```
[FunctionName("InspectionRouter")]
0 references
public static void Run([TimerTrigger("0 0 0 * * * ")]TimerInfo myTimer, ILogger log)
{
    log.LogInformation($"C# Timer trigger function executed at: {DateTime.Now}");

    CRMWebAPI api = GetCRMWebAPI(log).Result;
    dynamic whoami = api.ExecuteFunction("WhoAmI").Result;
    log.LogInformation($"UserID: {whoami.UserId}");

    var inspections = GetInspections(api).Result;
    var users = GetUsers(api).Result;
}
```

4. Create a method that will assign inspections to users

- Add the method below to the class.

```
private static async Task<CRMUpdateResult> RouteInspection(CRMWebAPI api, dynamic inspection, st
{
dynamic updateObject = new ExpandoObject();
((IDictionary<string, object>)updateObject).Add
("ownerid@odata.bind", "/systemusers(" + userId + ")");
updateObject.contoso_sequence = sequenceNumber.ToString();
return await api.Update("contoso_inspections", new Guid(inspection.contoso_inspectionid), update
}
```

5. Create two-digit random number.

- Add the code below to the Run method.

```
Random rnd = new Random();
int sequenceNumber = rnd.Next(10, 99);
```

6. Assign Inspections

- Go through the **Inspections** and call the **RouteInspection** method.

```
int currentUserIndex = 0;
foreach (dynamic inspection in inspections.List)
{
log.LogInformation($"Routing inspection {inspection.contoso_name}");
var inspectionResult = new CRMUpdateResult();
//Your record assignment would like this. We will not assign records to different users in this
// if (users.List.Count > (currentUserIndex))
//{
// dynamic currentUser = users.List[currentUserIndex];
// inspectionResult = RouteInspection(api, inspection, currentUser.systemuserid.ToString(), sequ
//currentUserIndex++;
//}
}
```

- We will not assign inspection records to other users in this lab. **Comment** out the **if** statement you just added, and we will be replacing it with logic to do the routing to our user only.

```
public static void Run([TimerTrigger("0 0 0 * * * ")]TimerInfo myTimer, ILogger log)
{
    log.LogInformation($"C# Timer trigger function executed at: {DateTime.Now}");

    CRMWebAPI api = GetCRMWebAPI(log).Result;
    dynamic whoami = api.ExecuteFunction("WhoAmI").Result;
    log.LogInformation($"UserID: {whoami.UserId}");

    var inspections = GetInspections(api).Result;
    var users = GetUsers(api).Result;

    Random rnd = new Random();
    int sequenceNumber = rnd.Next(10, 99);

    int currentUserIndex = 0;
    foreach (dynamic inspection in inspections.List)
    {
        log.LogInformation($"Routing inspection {inspection.contoso_name}");
        var inspectionResult = new CRMUpdateResult();
        //Your record assignment would like this. We will not assign records to different users in this lab
        //if (users.List.Count > (currentUserIndex))
        //{
        //    dynamic currentUser = users.List[currentUserIndex];
        //    inspectionResult = RouteInspection(api, inspection, currentUser.systemuserid.ToString(), sequenceNumber).Result;
        //    currentUserIndex++;
        //}
    }
}
```

- Assign inspections to the Inspection Router. Add the code below inside **foreach**.

```
//We will instead assign inspections to the user you are currently logged in as
inspectionResult = RouteInspection(api, inspection, whoami.UserId.ToString(), sequenceNumber).Res
```

```
0 references
public static void Run([TimerTrigger("0 0 0 * * * ")]TimerInfo myTimer, ILogger log)
{
    log.LogInformation($"C# Timer trigger function executed at: {DateTime.Now}");

    CRMWebAPI api = GetCRMWebAPI(log).Result;
    dynamic whoami = api.ExecuteFunction("WhoAmI").Result;
    log.LogInformation($"UserID: {whoami.UserId}");

    var inspections = GetInspections(api).Result;
    var users = GetUsers(api).Result;

    Random rnd = new Random();
    int sequenceNumber = rnd.Next(10, 99);

    int currentUserIndex = 0;
    foreach (dynamic inspection in inspections.List)
    {
        log.LogInformation($"Routing inspection {inspection.contoso_name}");
        var inspectionResult = new CRMUpdateResult();
        //Your record assignment would like this. We will not assign records to different users in this lab
        //if (users.List.Count > (currentUserIndex))
        //{
        //    dynamic currentUser = users.List[currentUserIndex];
        //    inspectionResult = RouteInspection(api, inspection, currentUser.systemuserid.ToString(), sequenceNumber).Result;
        //    currentUserIndex++;
        //}

        //We will instead assign inspections to the user you are currently logged in as
        inspectionResult = RouteInspection(api, inspection, whoami.UserId.ToString(), sequenceNumber).Result;
    }
}
```

Build the project and make sure that the build succeeds.

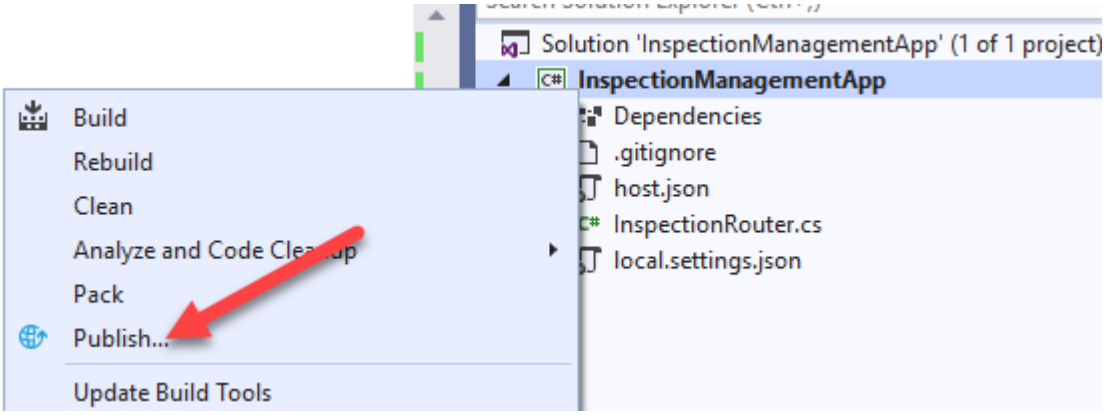
37 Exercise #3: Publish and Test

Objective: In this exercise, you will publish the Azure function to Azure, update the app settings, and test the function.

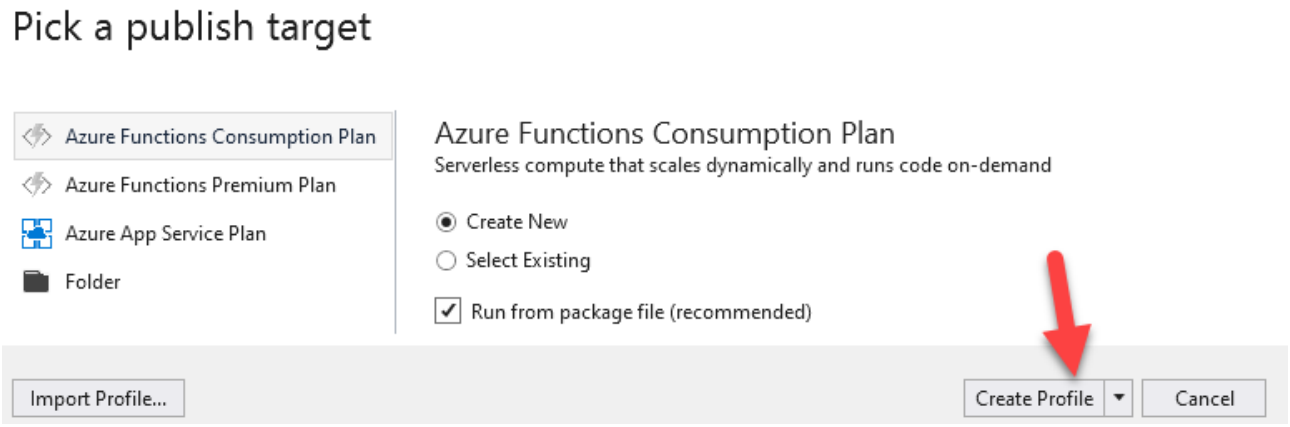
37.1 Task #1: Publish to Azure

1. Publish the function

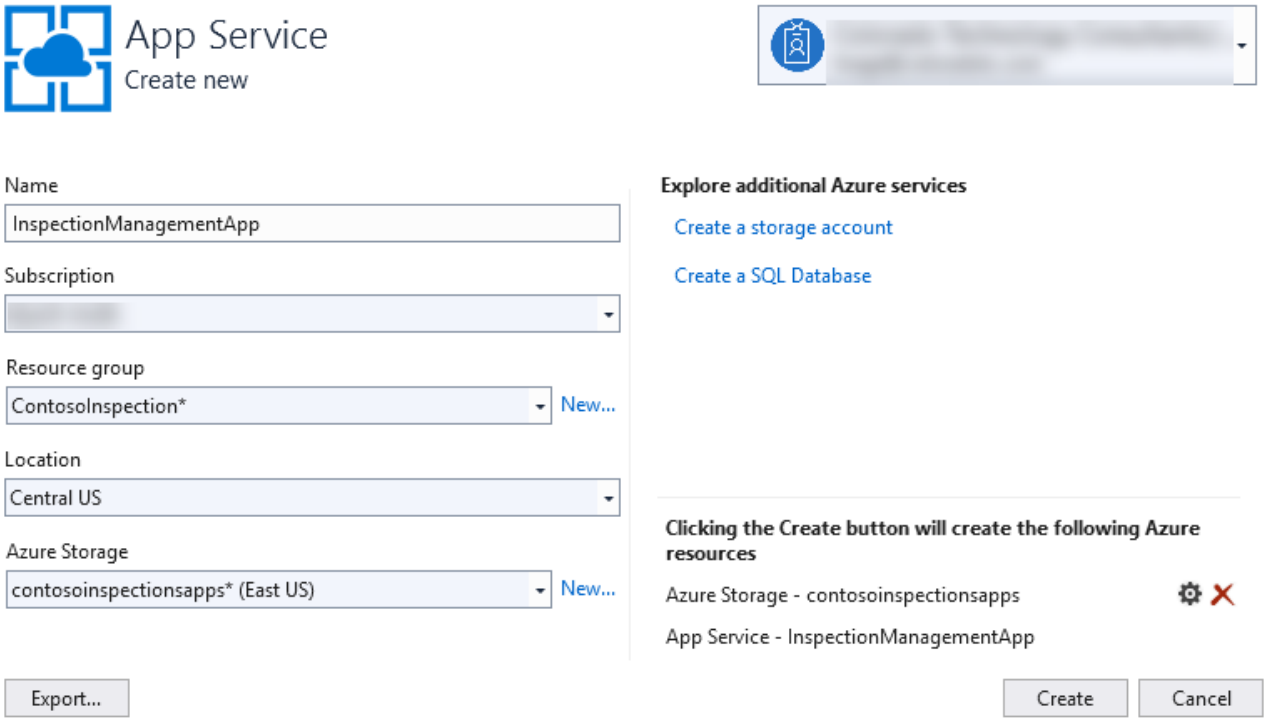
- Right click on the project and select **Publish**.



- Select **Create New** and click **Create Profile**.



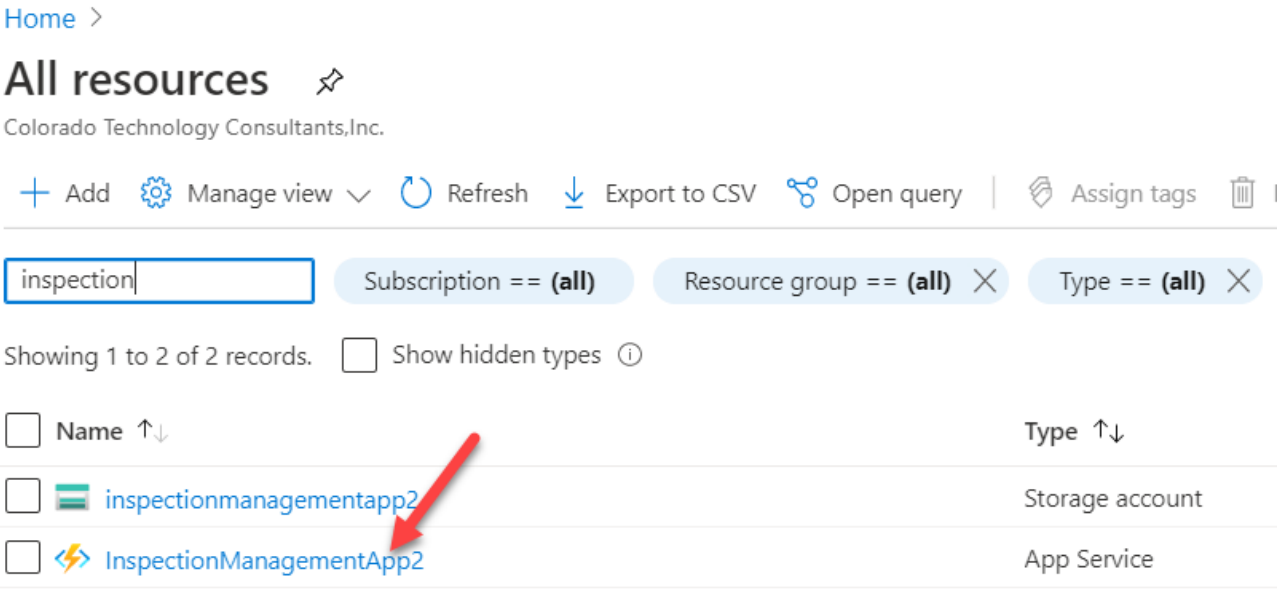
- Make sure you are logged in to your correct **Azure** account, enter a unique name for **App Name**, create **New Resource Group**, create **New Azure Storage**, and click **Create**.



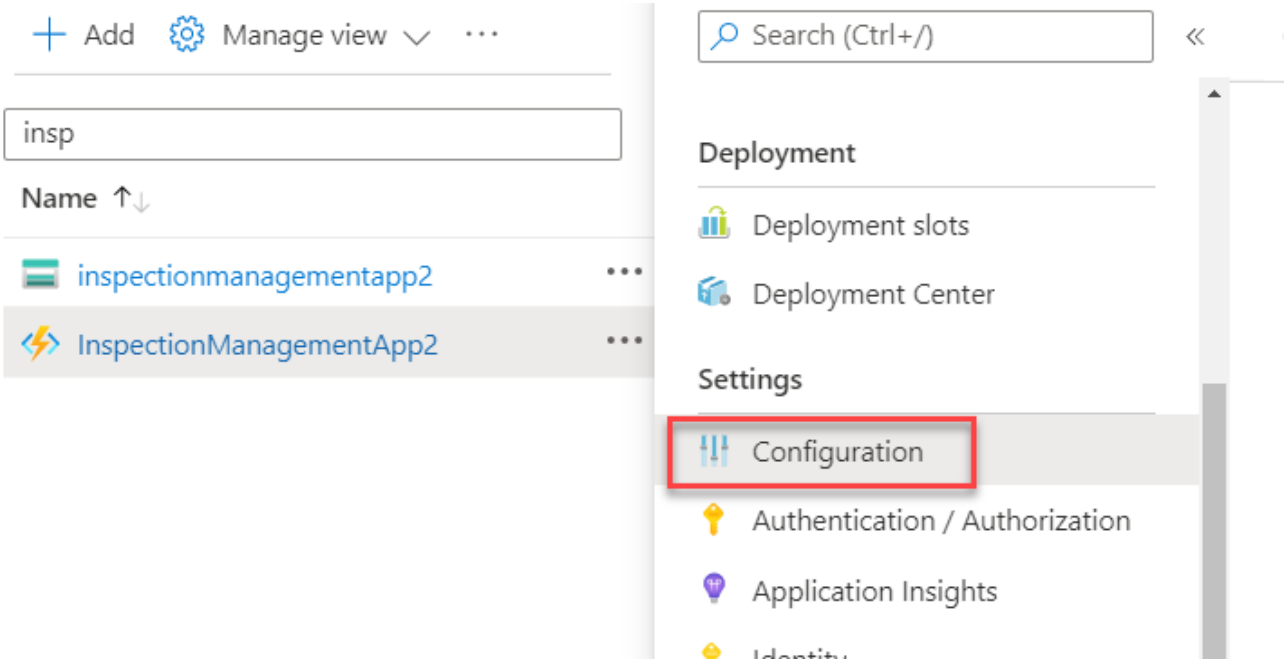
- Click **Publish**.
- Wait for the function application to be configured and published.

2. Open the function application settings

- Go back to you **Azure** portal.
- Select **All Resources**, search for **InspectionManagement**, and click to open the function you published.

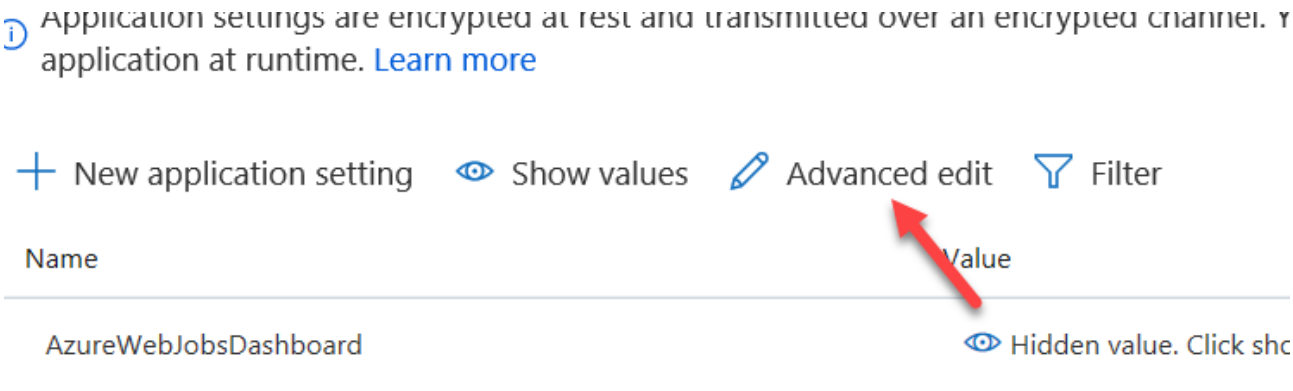


- Click scroll down to **Settings** and select **Configuration**.



3. Update App Settings

- Click **Advanced Edit**.



- Paste the json below at the top of the settings.

```
{
"name": "cdsclientid",
"value": "[clientid]",
"slotSetting": false
},
{
"name": "cdsclientsecret",
"value": "[clientsecret]",
"slotSetting": false
},
{
"name": "cdsurl",
"value": "[cdsurl]",
"slotSetting": false
},
```

```
[
  {
    "name": "cdsclientid",
    "value": "[clientid]",
    "slotSetting": false
  },
  {
    "name": "cdsclientsecret",
    "value": "[clientsecret]",
    "slotSetting": false
  },
  {
    "name": "cdsurl",
    "value": "[cdsurl]",
    "slotSetting": false
  },
  {
    "name": "AzureWebJobsDashboard",
    "value": "DefaultEndpointsProtocol=https;AccountName=contosoinspectionsapps;Ac
    "slotSetting": false
  },
  {
    "name": "AzureWebJobsStorage",
```

- Go back to **Visual Studio** and open the **local.settings.json** file.
- You will copy the **cdsurl**, **cdsclientid**, and **cdsclientsecret**. Copy the **cdsurl** value.

```
{
  "IsEncrypted": false,
  "Values": {
    "AzureWebJobsStorage": "UseDevelopmentStorage=true",
    "FUNCTIONS_WORKER_RUNTIME": "dotnet",
    "cdsurl": "https://orga5253b09.crm.dynamics.com",
    "cdsclientid": "b1513bf1-5d08-43d3-8507-48a646536d28",
    "cdsclientsecret": "8qYOPEwy4lNu--bd[G/KY6aVp0gXQQx3"
  }
}
```

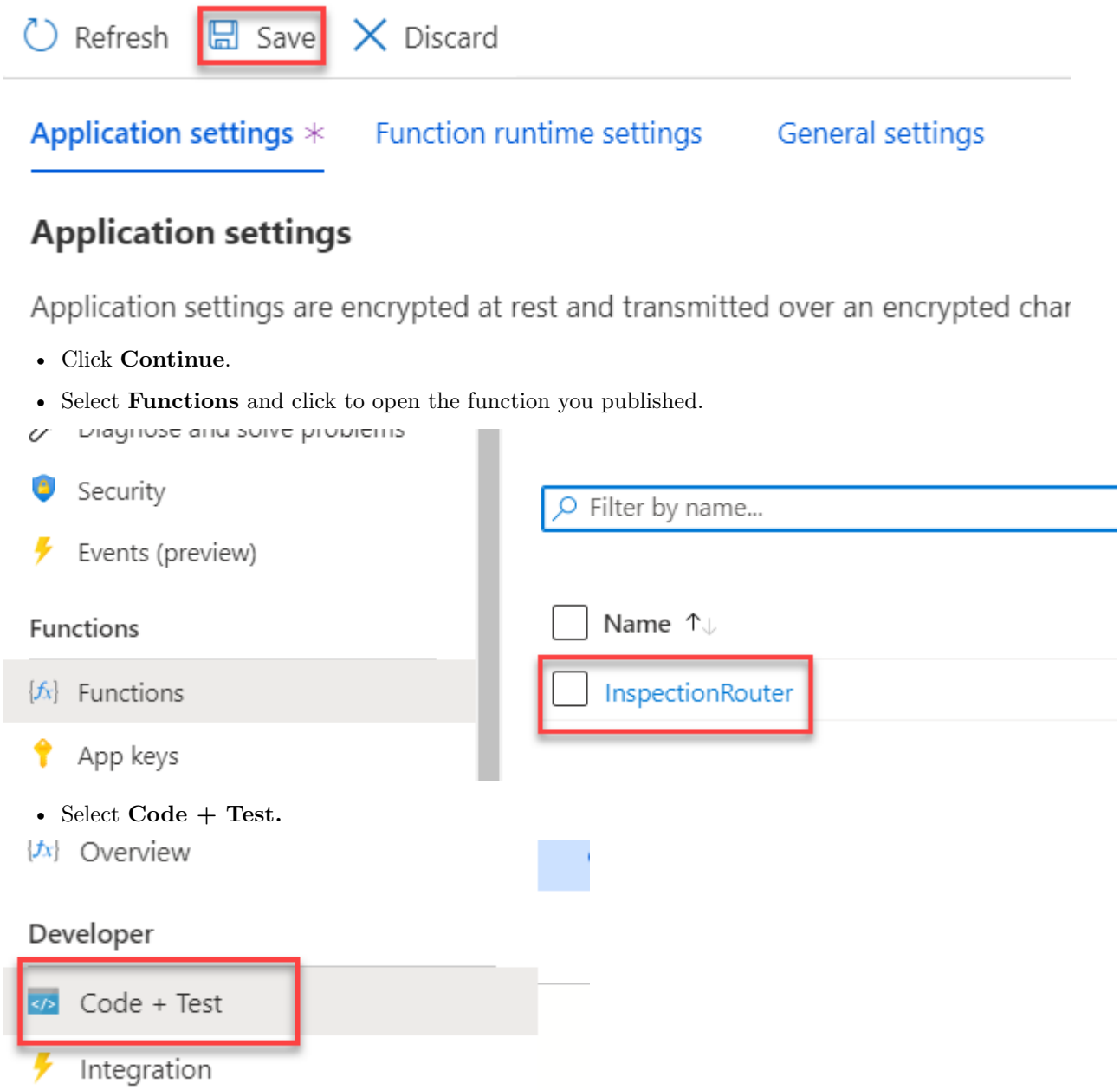
- Go back to **Azure** and replace [**cdsurl**] with the URL you copied.

```
[
  {
    "name": "cdsclientid",
    "value": "[clientid]",
    "slotSetting": false
  },
  {
    "name": "cdsclientsecret",
    "value": "[clientsecret]",
    "slotSetting": false
  },
  {
    "name": "cdsurl",
    "value": "https://orga5253b09.crm.dynamics.com",
    "slotSetting": false
  },
  {
    "name": "AzureWebJobsDashboard",
```


- Copy the **cdsclientid** and **cdsclientsecret** values from the **local.settings.json** file and replace [**cdsclientid**] and [**cdsclientsecret**].


```
[
  {
    "name": "cdsclientid",
    "value": "b1513bf1-5d08-43d3-8507-48a646536d28",
    "slotSetting": false
  },
  {
    "name": "cdsclientsecret",
    "value": "8qYOPEwy4lNu--bd[G/KY6aVp0gXQQx3",
    "slotSetting": false
  },
  {
    "name": "cdsurl",
    "value": "https://orga5253b09.crm.dynamics.com",
    "slotSetting": false
  },
  {
    "name": "AzureWebJobsDashboard".
```

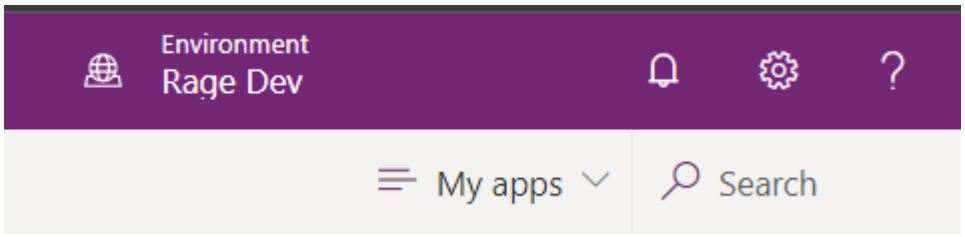

- Click **OK**.
- Click **Save**.



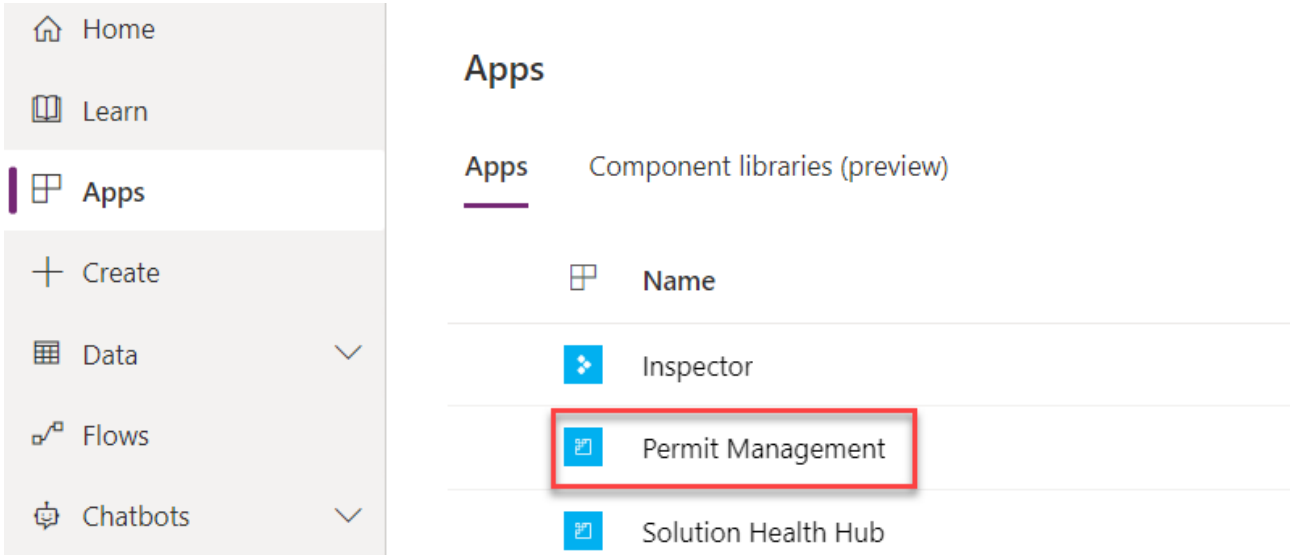
- Click **Continue**.
- Select **Functions** and click to open the function you published.
- Select **Code + Test.**

4. Prepare test record

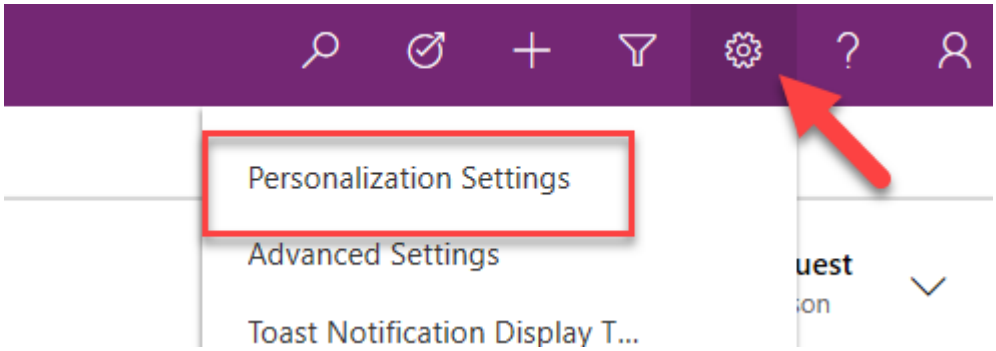
- Start a new browser window and sign in to Power Apps maker portal
- Make sure you are in the **Dev** environment.



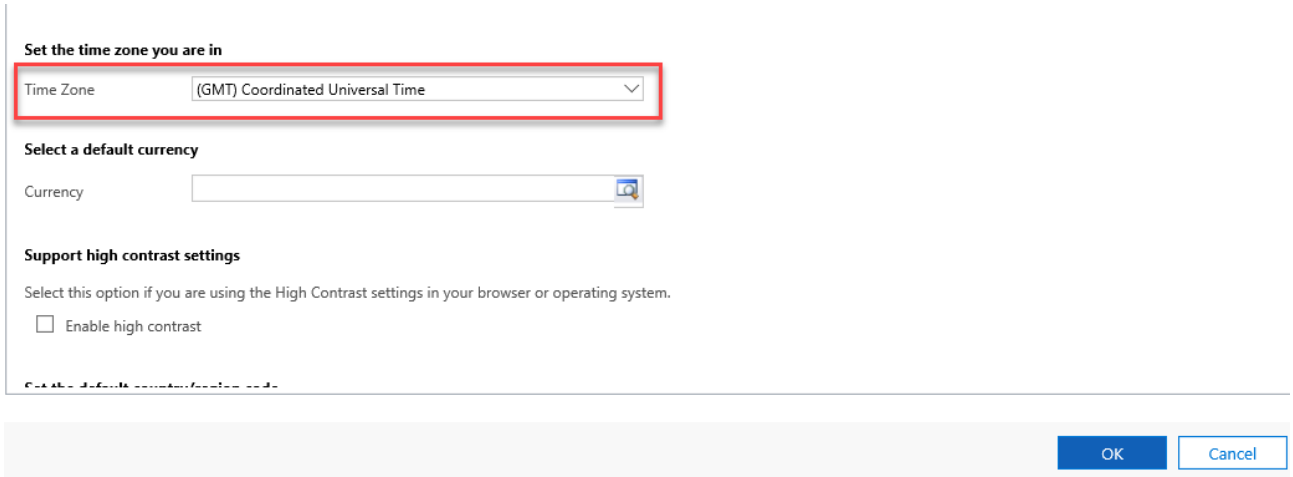
- Select **Apps** and click to open the **Permit Management** application.



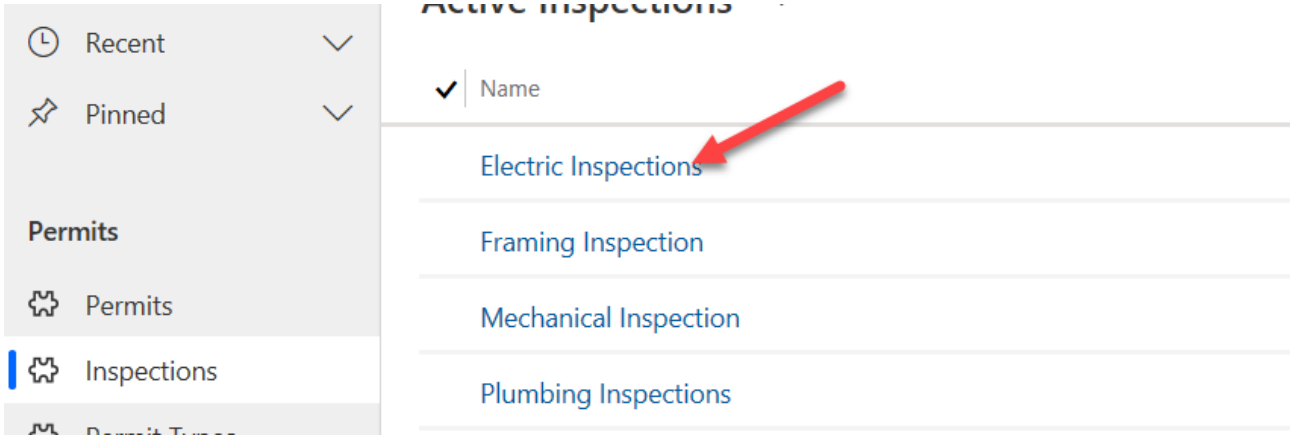
- Click Settings and select **Personalization and** Settings.



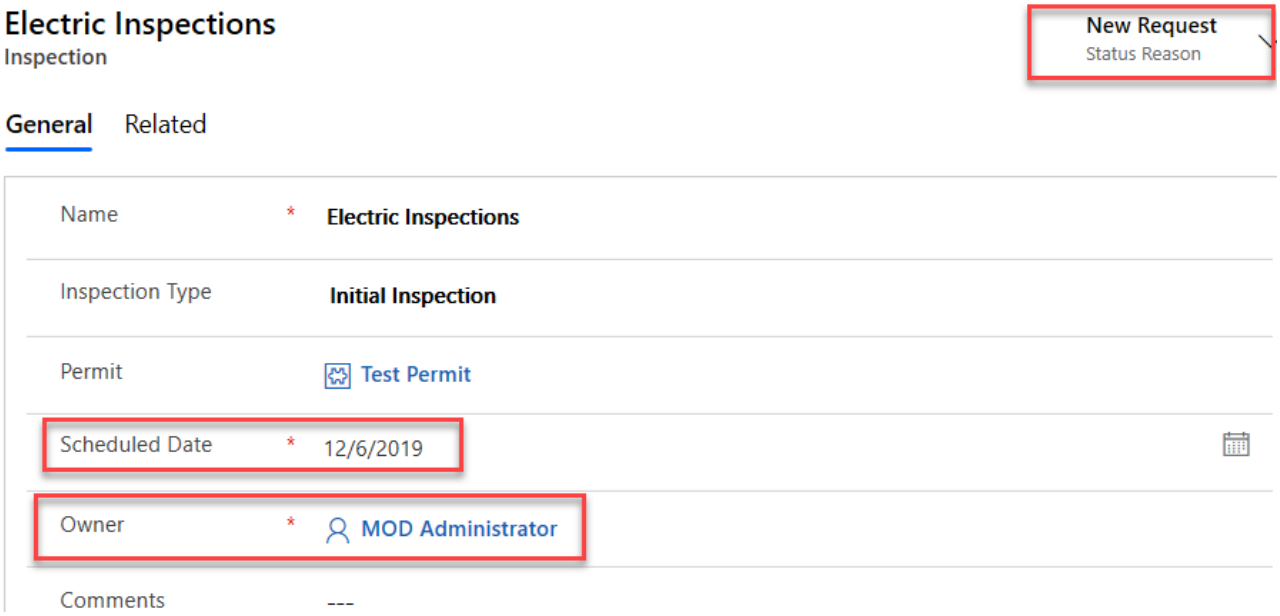
- Change the **Time Zone** to **(GMT-11:00) Coordinated Universal Time-11** and click **OK**. This will ensure the query results will produce the same results regardless of your time zone.



- Select **Inspections** and click to open one of the **Inspection** records or create a new record.



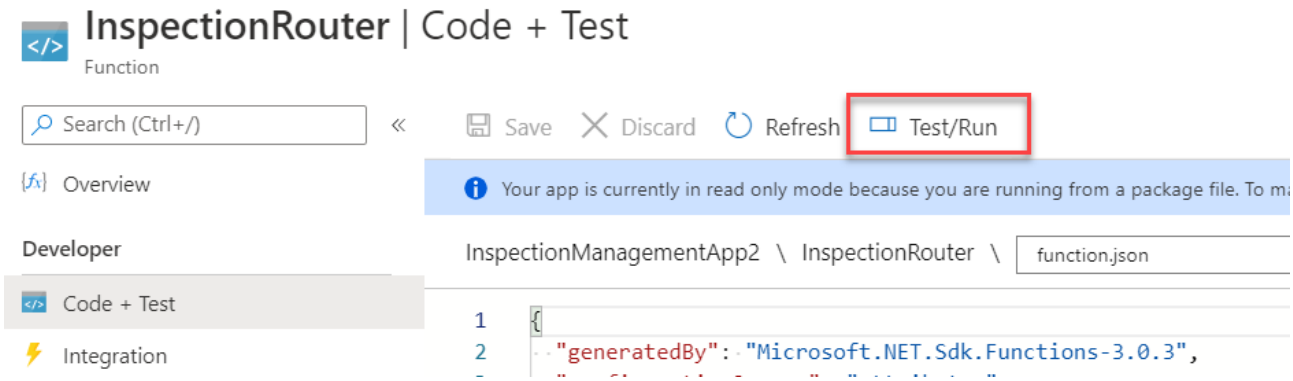
- Set the **Status Reason** to **New Request** or **Pending**, change the **Scheduled Date** to today's date, and make a note of the current **Owner** of the record.



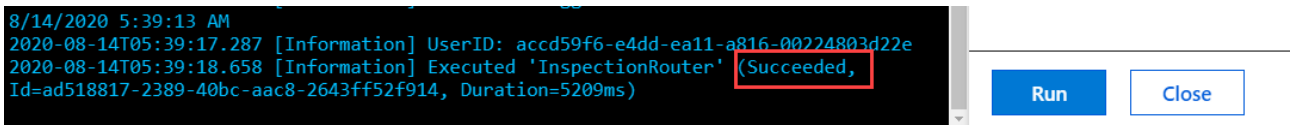
- Click **Save**.

5. Run the function

- Go to your **Azure** portal.
- Click **Test/Run.**

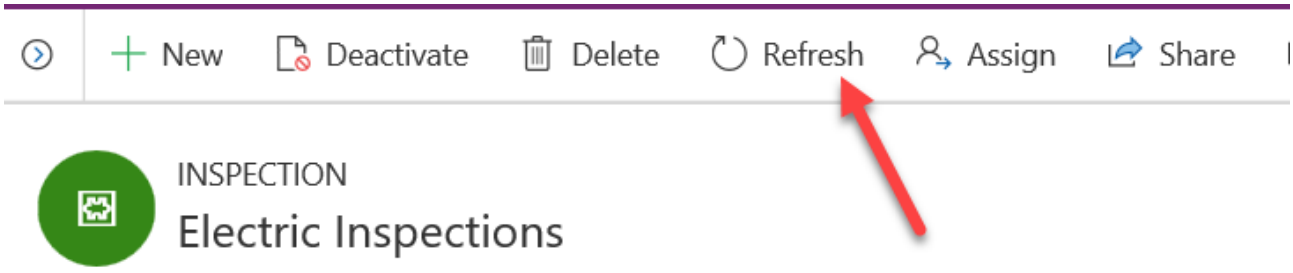


- Click **Run**.
- The function should run and succeed.

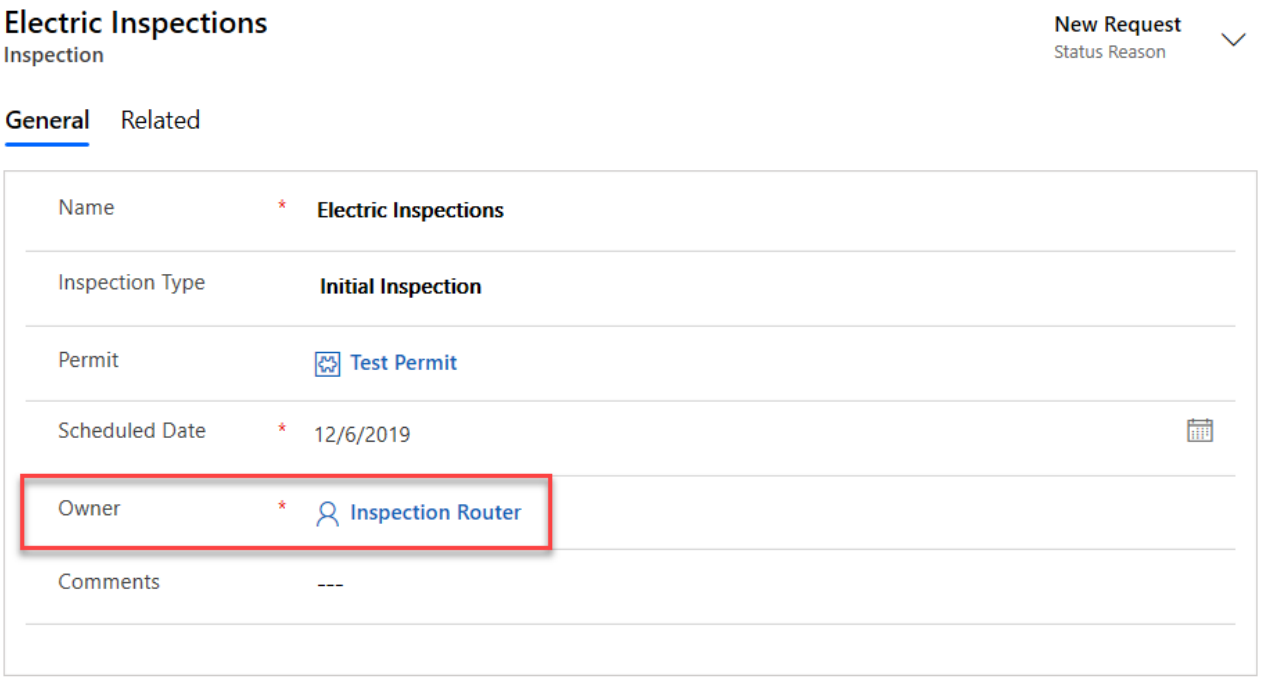


6. Confirm record assignment

- Go back to the **Permit Management** application.
- Click **Refresh**.



- The record **Owner** should now be the **Inspection Router**.



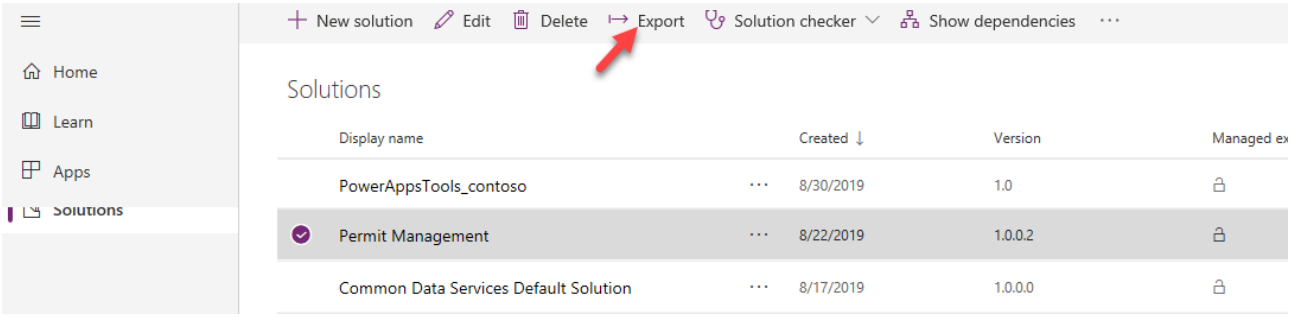
38 Exercise #4: Promote to production

Objective: In this exercise, you will export the Permit Management solution form your Dev environment and import it into your Production environment. In this lab, you have added a security role to the solution that must be promoted.

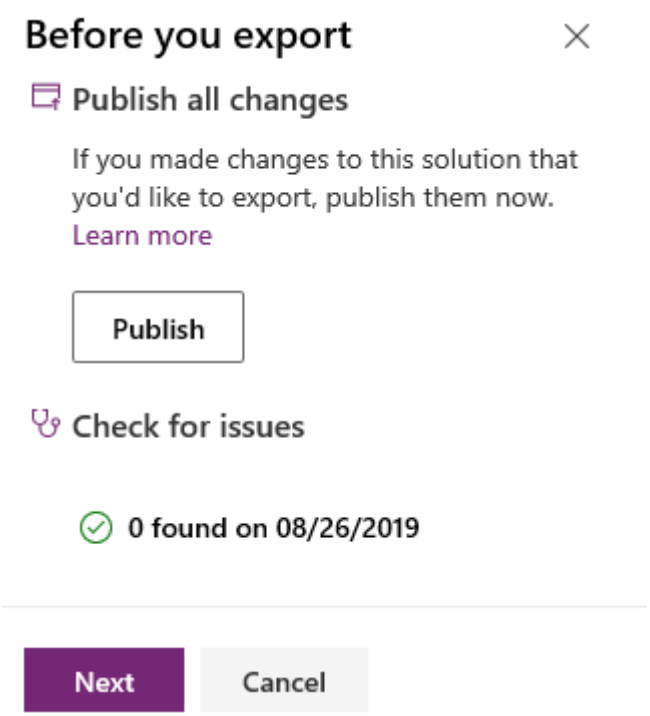
38.1 Task #1: Export Solution

1. Export Permit Management managed solution

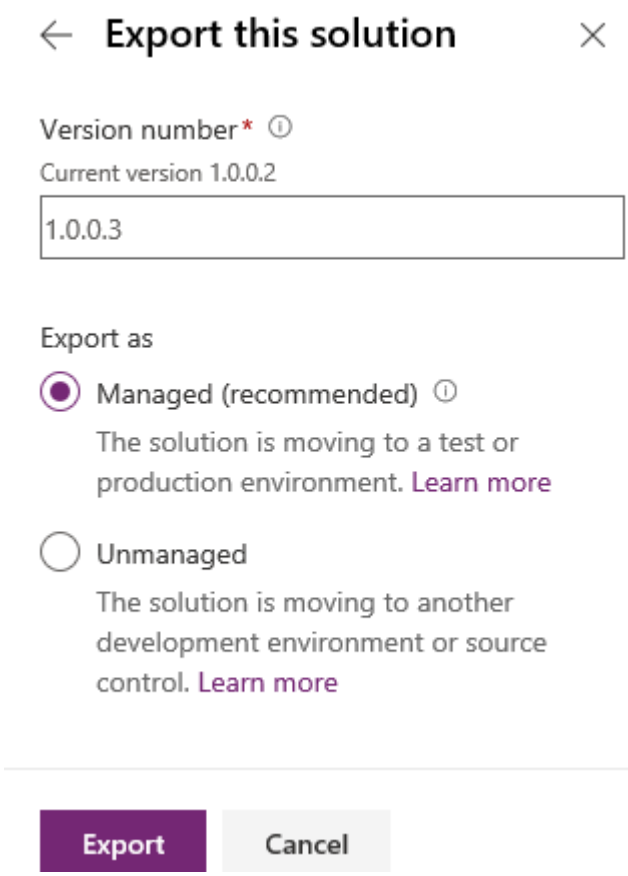
- Sign in to Power Apps maker portal and make sure you are in the **Dev** environment.
- Select **Solution**.
- Select the **Permit Management** solution and click **Export**.



- Click **Publish** and wait for the publishing to complete.



- Click **Next**.
- Select **Managed** and click **Export**.



- Save the **Exported** solution on your machine.

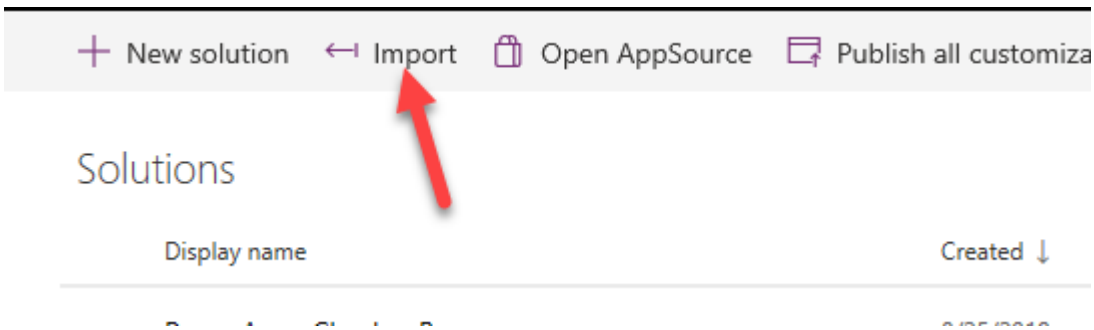
2. Export Permit Management unmanaged solution

- Select **Solution** again.
- Select the **Permit Management** solution and click **Export**.
- Click **Next**.
- Select **Unmanaged, edit the version number to match the Managed solution** and click **Export**.
- Save the **Exported** solution in your machine.

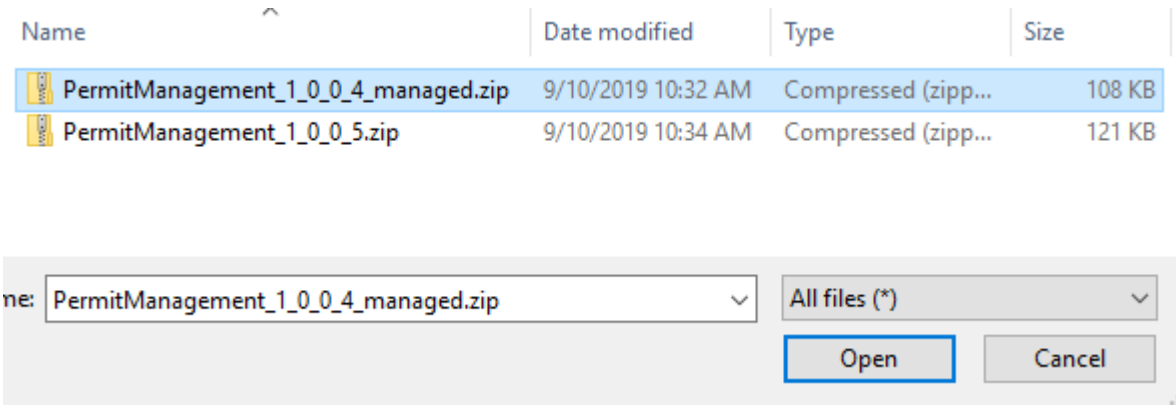
38.2 Task #2: Import Solution

1. Import Permit Management managed solution

- Sign in to https://make.powerapps.com and make sure you are in the **Prod** environment.
- Select **Solution**.
- Click **Import**.



- Click **Choose file**.
- Select the **Managed** solution you exported and click **Open**.



- Click **Next**.
- Click **Next** again.
- Click **Import**.
- Wait for the import to complete and click **Close**.
- Review and test your production environment.

38.3 lab: title: 'Lab 08: Publishing Events Externally'

[!NOTE] Effective November 2020:

- Common Data Service has been renamed to Microsoft Dataverse. Learn more
- Some terminology in Microsoft Dataverse has been updated. For example, *entity* is now *table* and *Column* is now *column*. Learn more

This content will be updated soon to reflect the latest terminology.

38.4 Lab 08 – Publishing Events Externally

39 Scenario

A regional building department issues and tracks permits for new buildings and updates for remodeling of existing buildings. Throughout this course you will build applications and automation to enable the regional building department to manage the permitting process. This will be an end-to-end solution which will help you understand the overall process flow.

In this lab you will use the event publishing capability of the Microsoft Dataverse. When a permit results in changing the size of the build site, an external taxing authority needs to be notified so they can evaluate if additional taxing is required. You will configure Microsoft Dataverse to publish permits with size changes using the web hook option. To simulate the taxing authority receiving the information you will create a simple Azure function to receive the post.

40 High-level lab steps

As part of configuring the event publishing, you will complete the following:

- Create an Azure Function to receive the web hook post
- Configure Microsoft Dataverse to publish events using a web hook
- Test publishing of events

40.1 Things to consider before you begin

- Do we know what events will trigger our web hook?
- Could what we are doing with the web hook, be done using Power Automate?

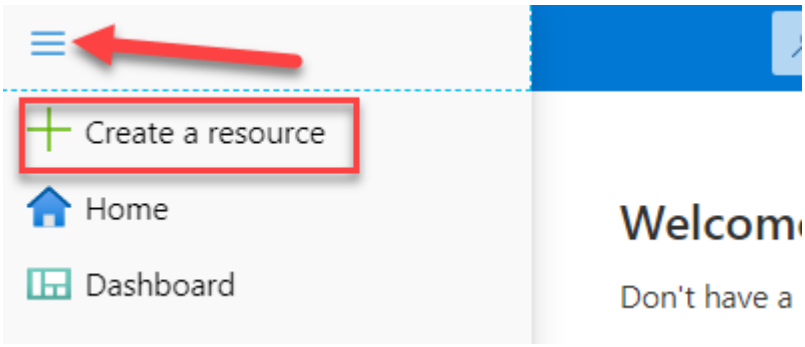
- Remember to continue working in your DEVELOPMENT environment. We'll move everything to production soon.

41 Exercise #1: Create an Azure Function

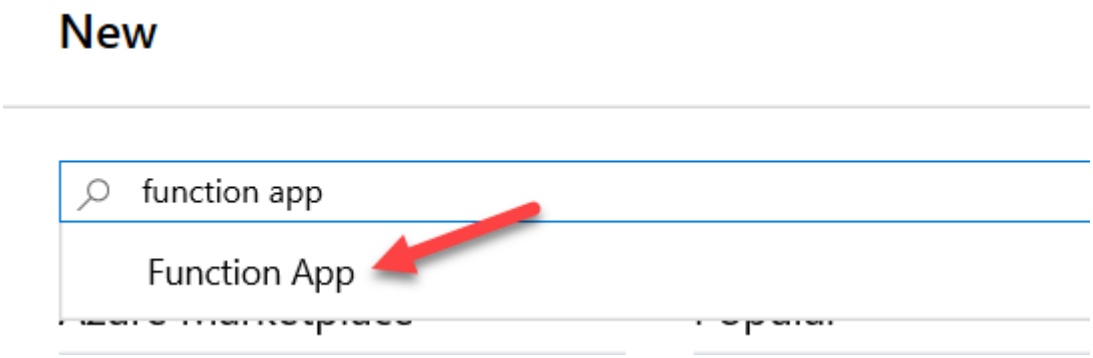
Objective: In this exercise, you will create an Azure Function that will be the endpoint to accept and log incoming web requests.

41.1 Task #1: Create Azure Function App

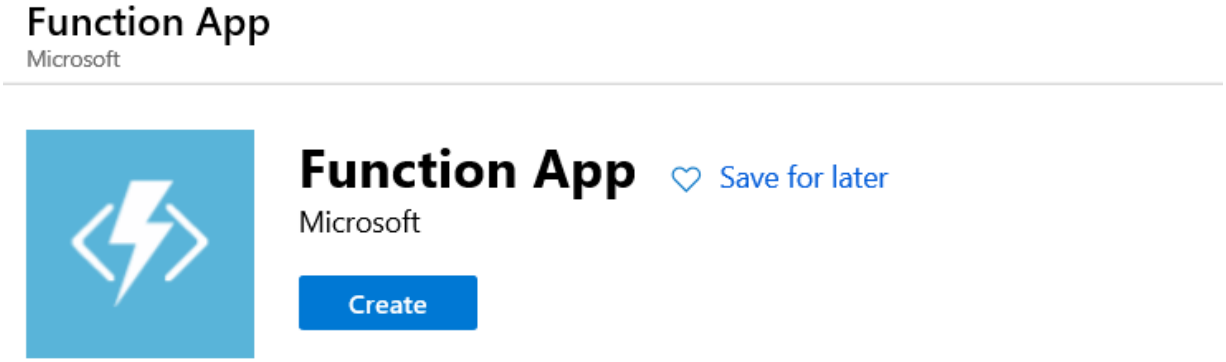
1. Create new function application
   - Sign in to Azure portal and login.
   - Click **Show portal menu** and select + **Create a Resource**.



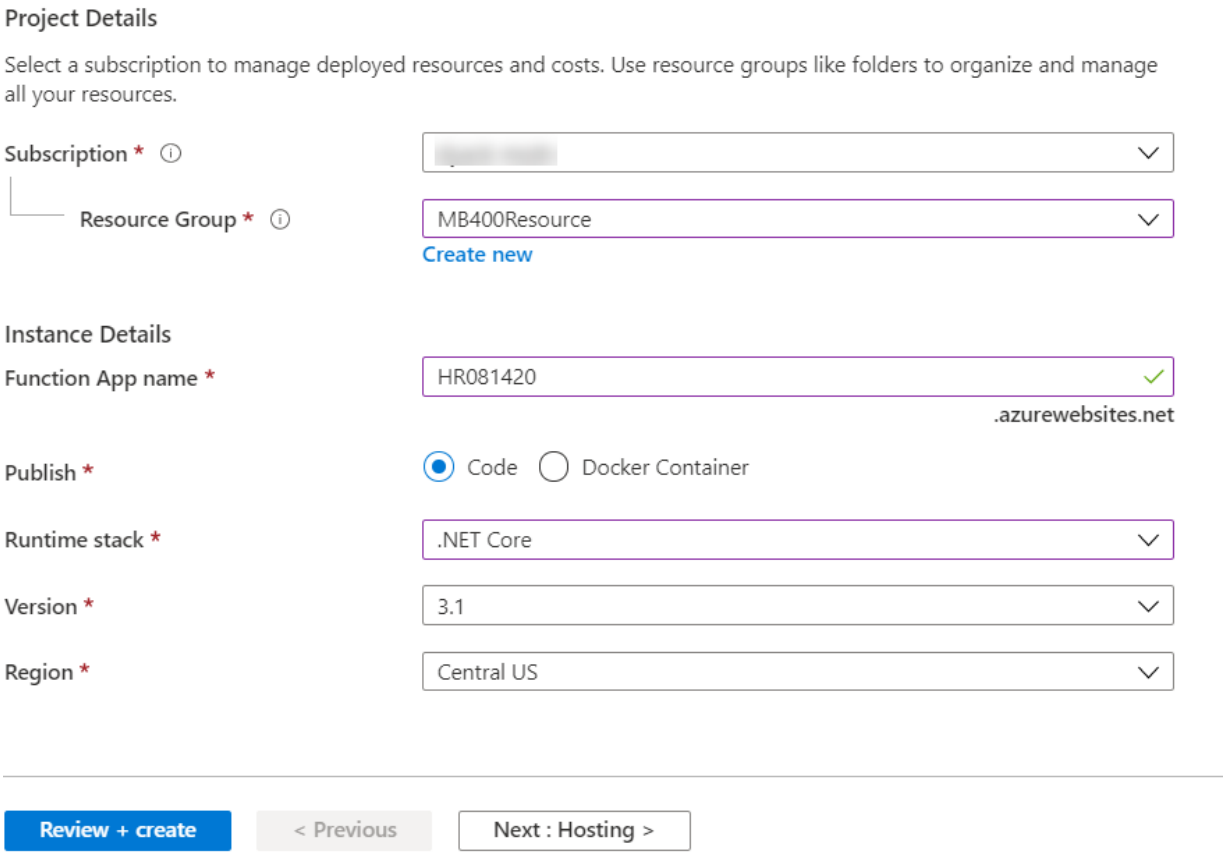
   - Search for Function App and select it.



   - Click **Create**.



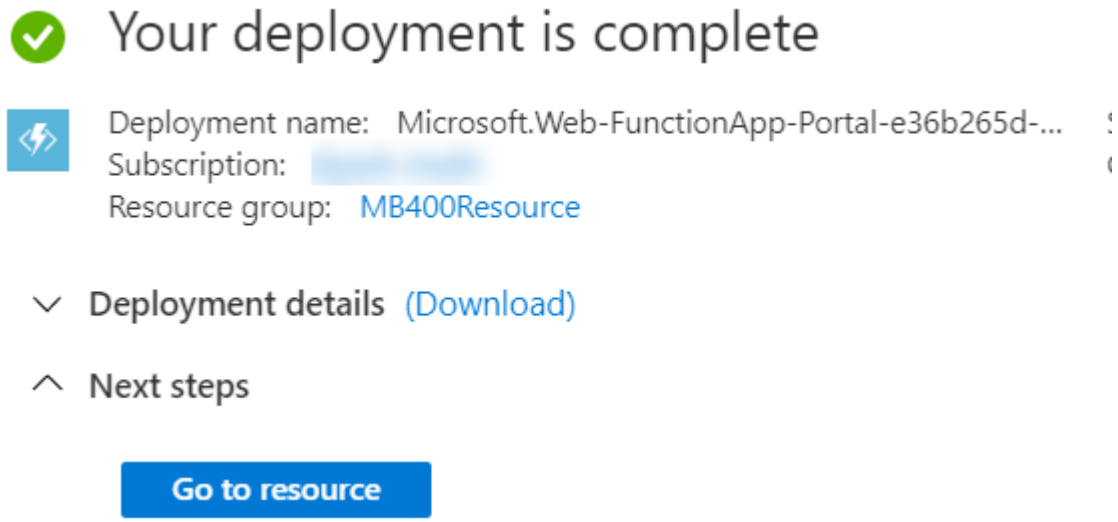
   - Enter your initials plus today's date for **App Name**, select your **Subscription**, select **Create New** for **Resource Group**, select **.NET Core** for Runtime Stack, select location in the same region as **Microsoft Dataverse**, and click **Review + Create**.



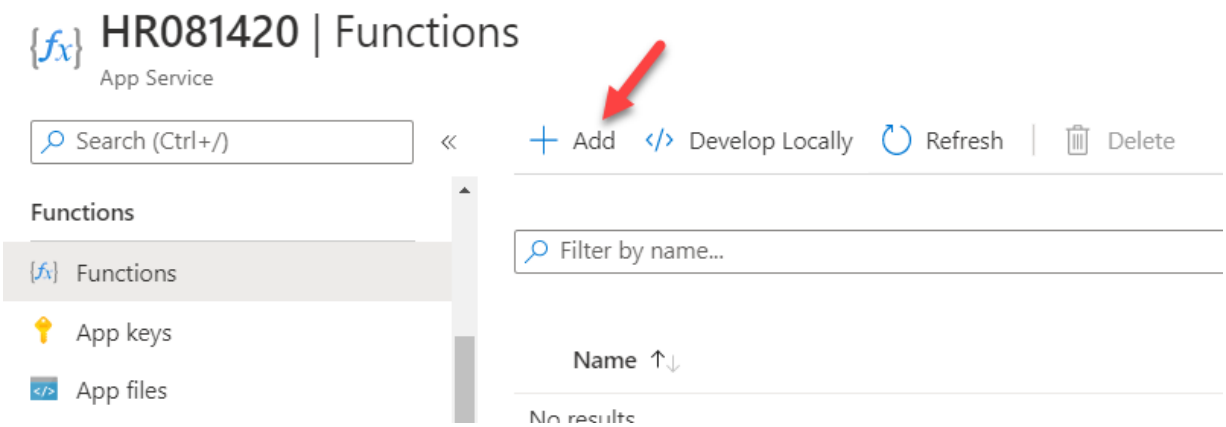
- Click **Create** and wait for the deployment to complete.

41.2 Task #2: Create an Azure Function

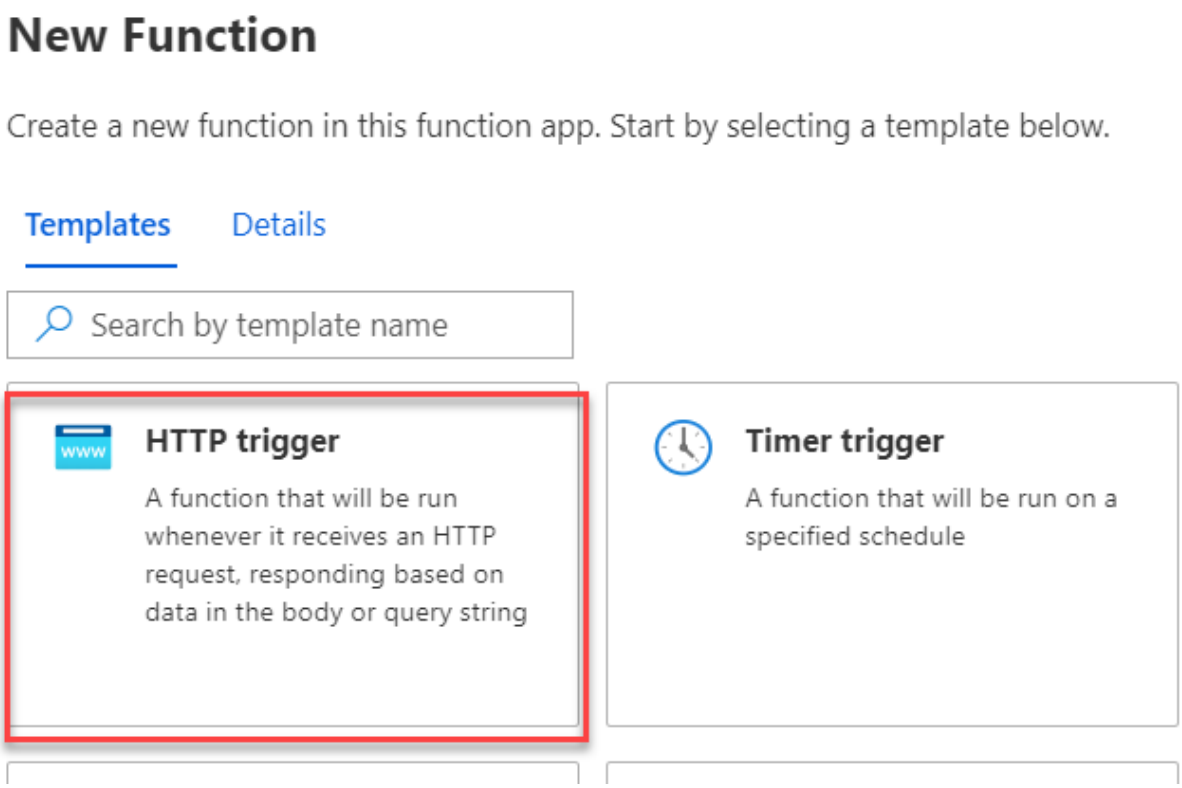
1. Create a new function
   - Click **Go to resource**.



   - Select **Functions** and click **+ Add**.

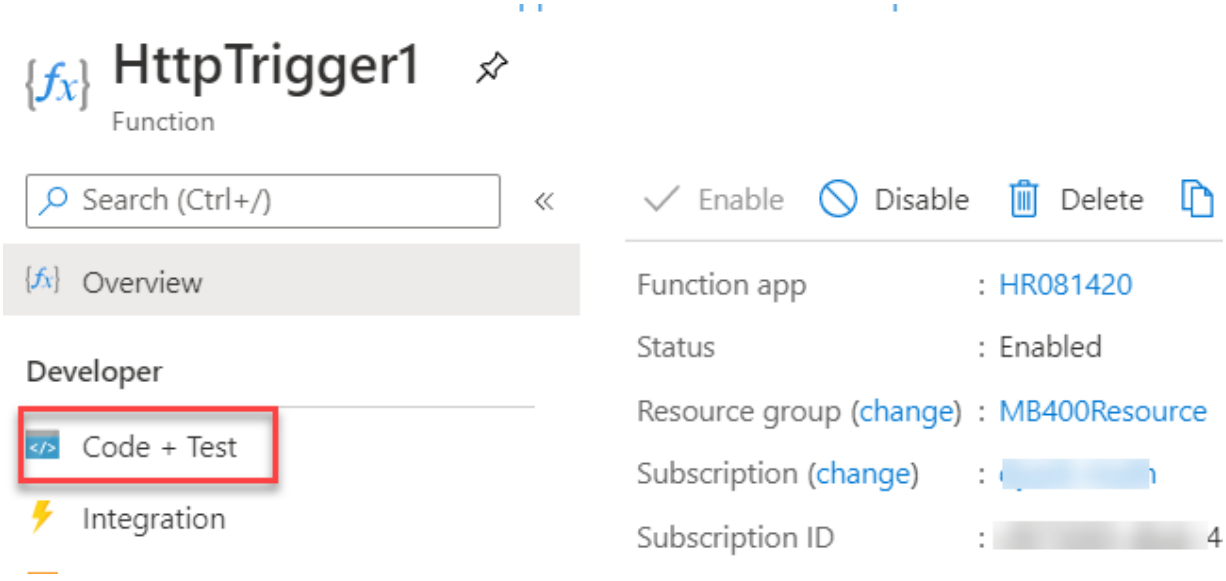


- Select **HTTP trigger**.

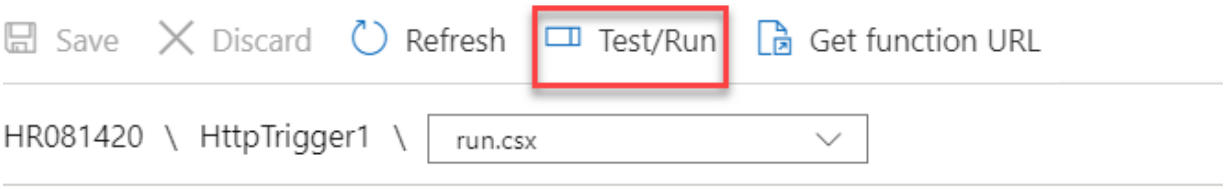


- Click **Create Function** and wait for the function to be created.

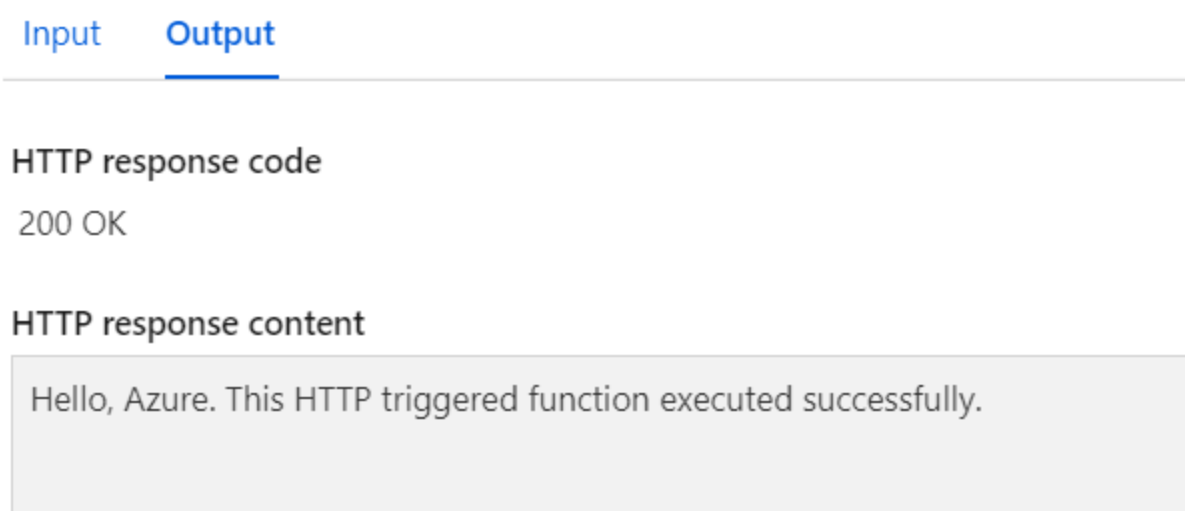
2. Test the function
   - Select **Code + Test**.



- Click **Test/Run**.



- Click **Run**.
- You should see **Hello, Azure** in the output.



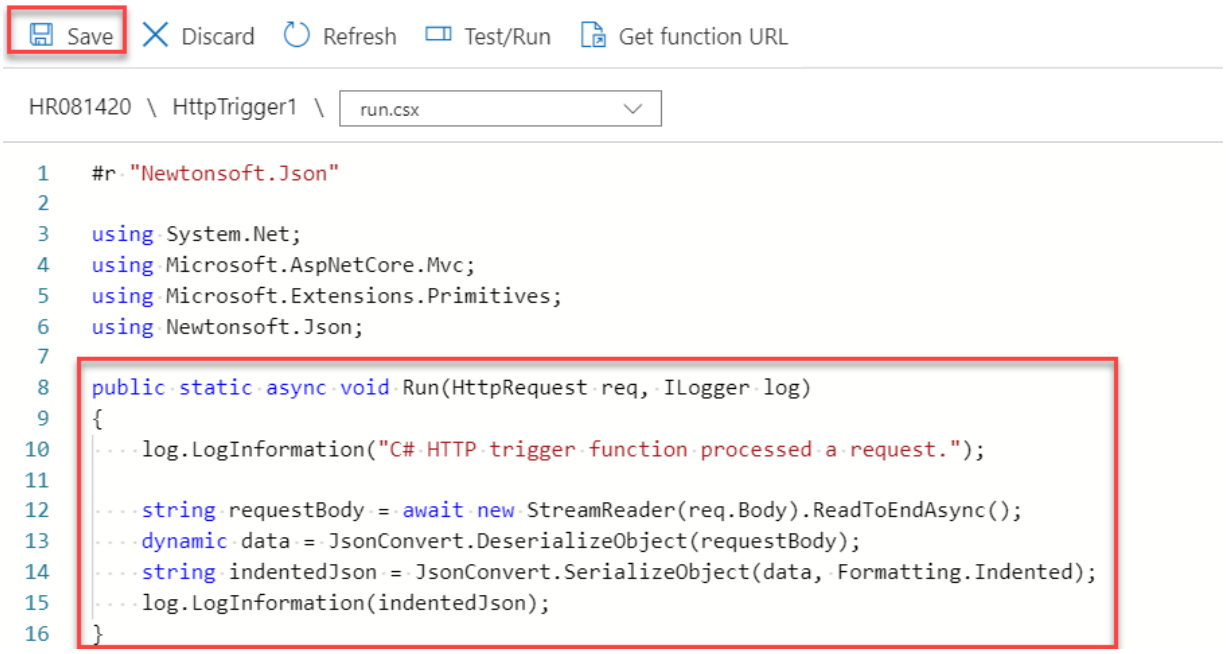
- Close the test pane.

3. Edit the function
   - Replace the Run method with the method below.

```
public static async void Run(HttpRequest req, ILogger log)
{
    log.LogInformation("C# HTTP trigger function processed a request.");

    string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
    dynamic data = JsonConvert.DeserializeObject(requestBody);
    string indentedJson = JsonConvert.SerializeObject(data, Formatting.Indented);
    log.LogInformation(indentedJson);
```

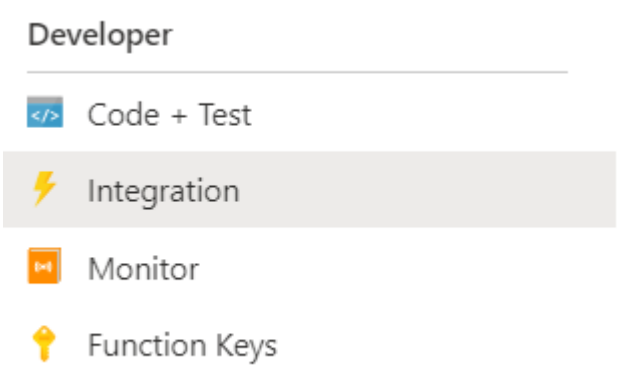
```
}
```

- Save your changes.

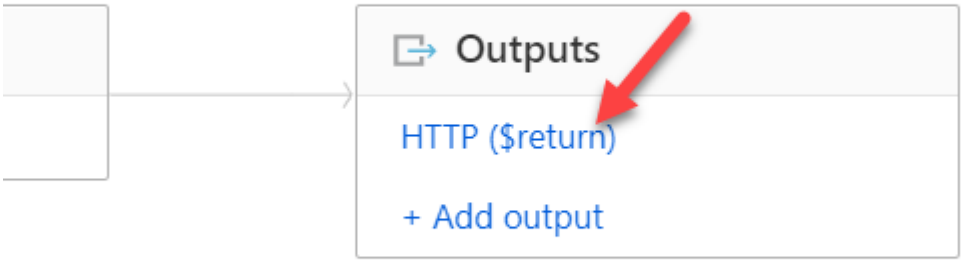


4. Remove HTTP output

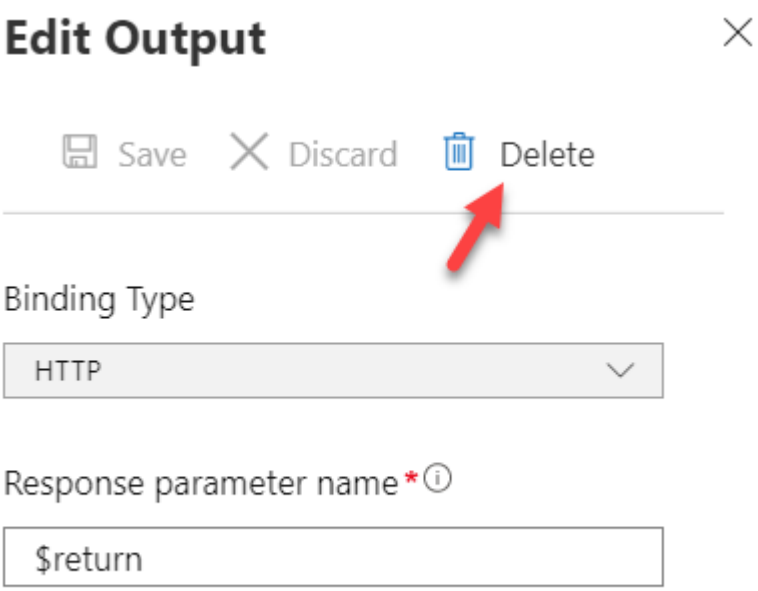
- Select **Integration**.



- Select the **HTTP Output**.

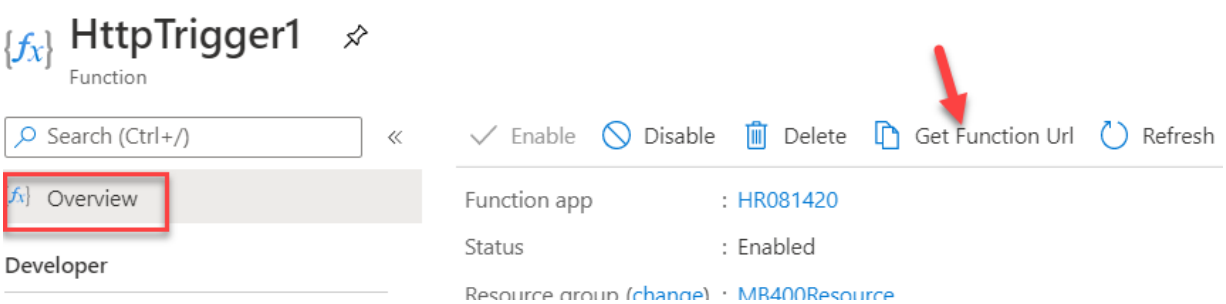


- Click **Delete**.

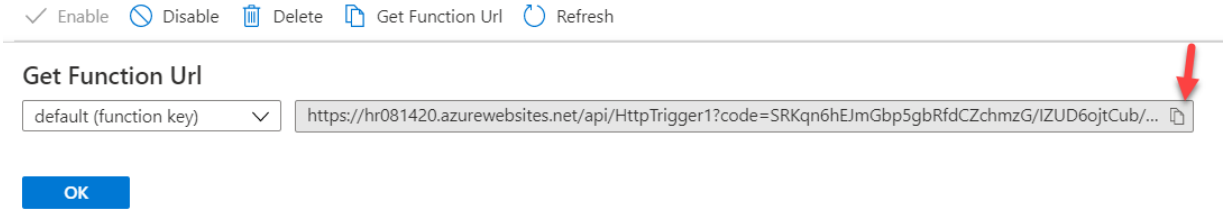


- Click **OK**.

5. Get the function URL
   - Select **Overview** and click **Get Function URL**.



   - Click **Copy** and click OK to close the popup.

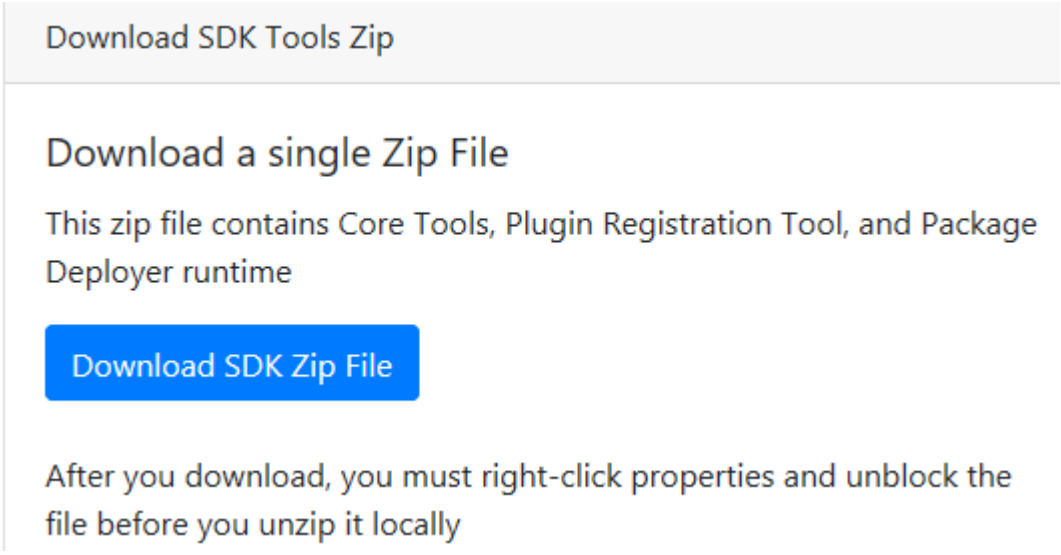


- Save the **URL**, you will need it in the next exercise.

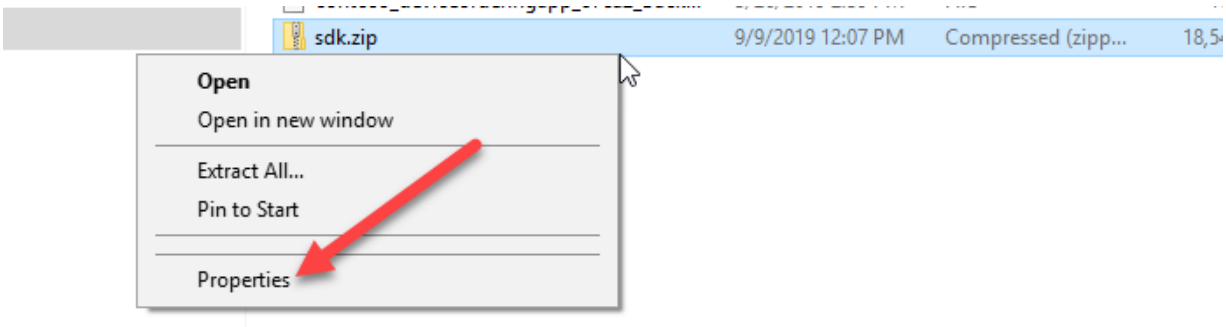
42 Exercise #2: Configure Web Hook

42.1 Task #1: Configure publishing to a web hook

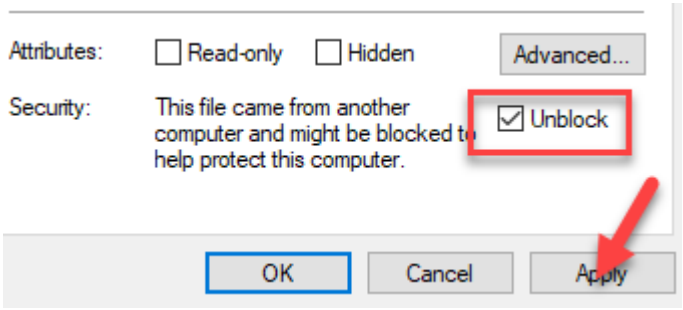
1. Download the SDK Toolkit. If you already have the Plugin Registration tool from the previous lab you can proceed to step three of this task.
   - Navigate to https://xrm.tools/SDK
   - Click **Download SDK Zip File**.



   - Save the zip file on your machine.
   - Right click on the downloaded **sdk.zip** file and select **Properties**.

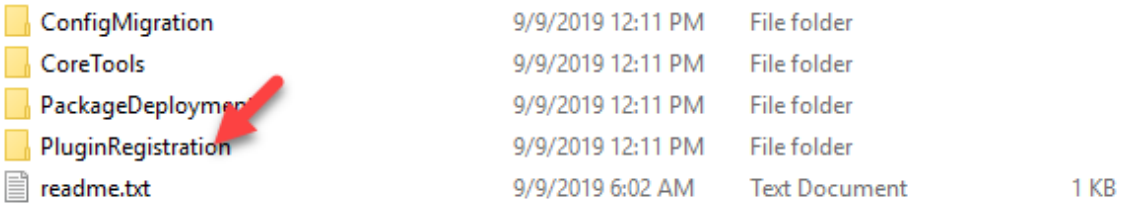


   - Check the **Unblock** checkbox and click Apply.

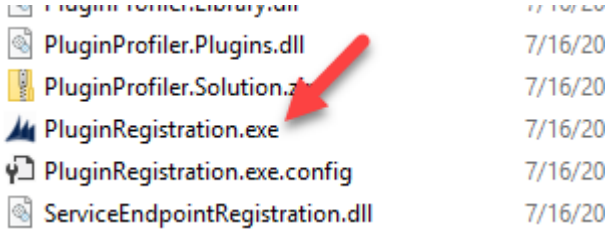


- Click **OK**.
- Right click on the **sdk.zip** file again and select **Extract All**.
- Complete extracting.

2. Start the Plugin Registration Tool
   - Open the **sdk** folder you extracted and click to open the **PluginRegistration** folder.

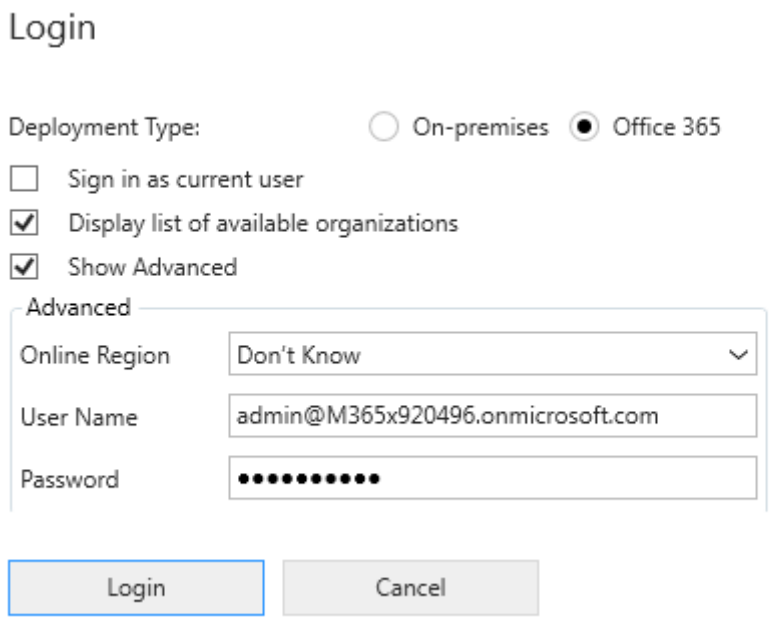


   - Locate and double click PluginRegistration.exe.

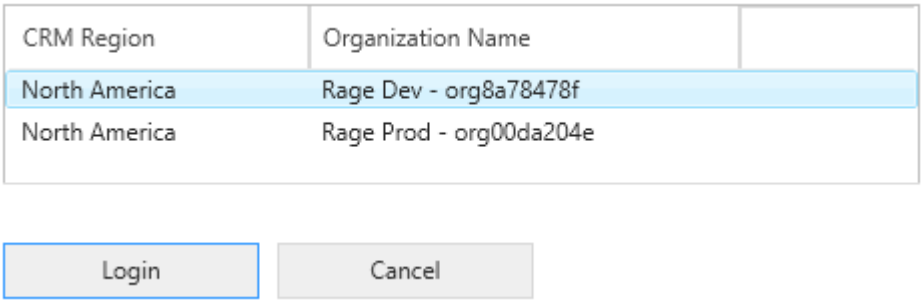


3. Create new connection
   - Click **Create New Connection**.

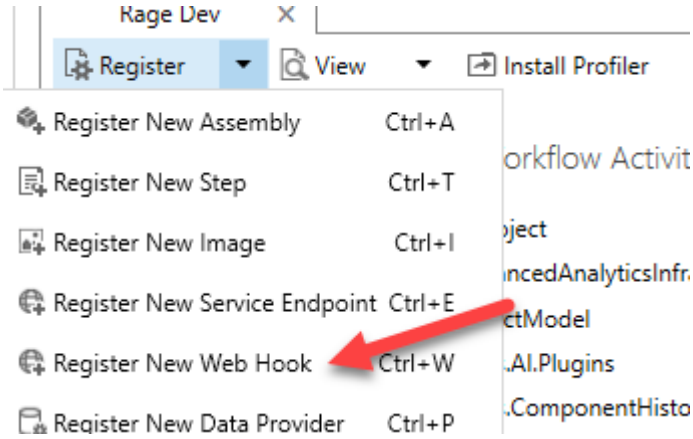
   - Select **Office 365** and check the **Display List of available organization** and **Show Advanced** checkboxes. Select **Online Region** where your organization is located. If you are unsure what region to select, select **Don't Know**.
   - Provide your **Microsoft Dataverse** credentials and click **Login**.



- Select the **Dev** environment and click **Login**.



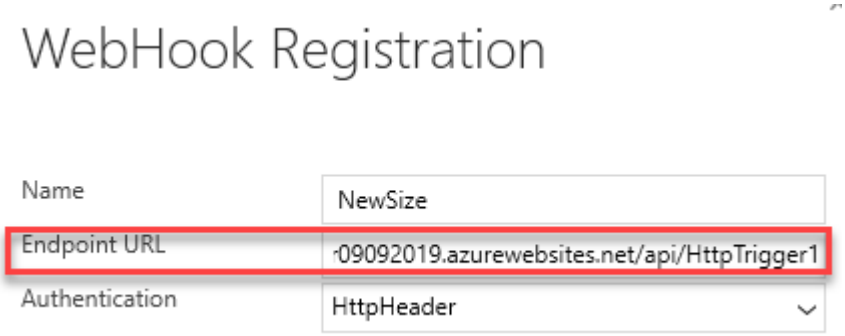
4. Register new Web Hook
   - Click **Register** and select **Register New Web Hook**.



   - Enter **NewSize** for **Name**.
   - Go to the notepad where you saved the function URL and copy everything before the '**?**'.

```
https://hr09092019.azurewebsites.net/api/HttpTrigger1?
code=ecVN3/JiBMDljpAn8W/xk1/oNose4NUFs14oexaBI9vasKcw3uksvg==
```

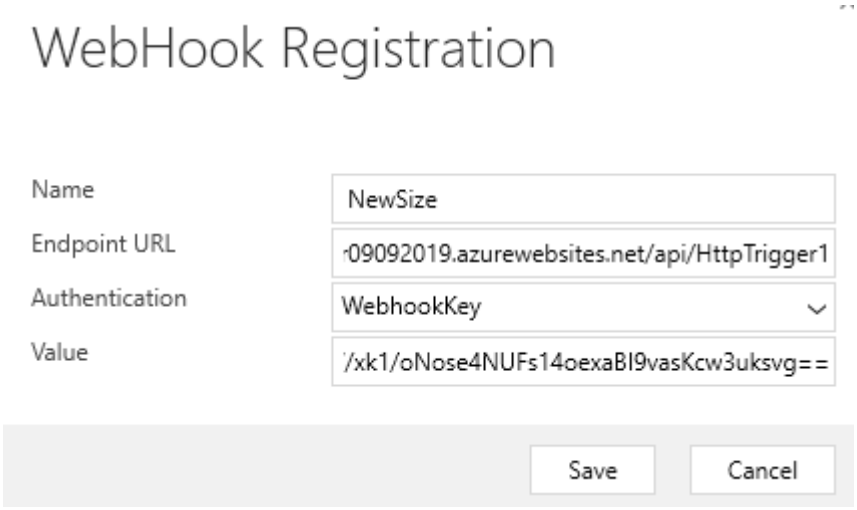
- Go back to the **Plugin Registration** tool and paste the **URL** you copied in the **Endpoint URL** Column.



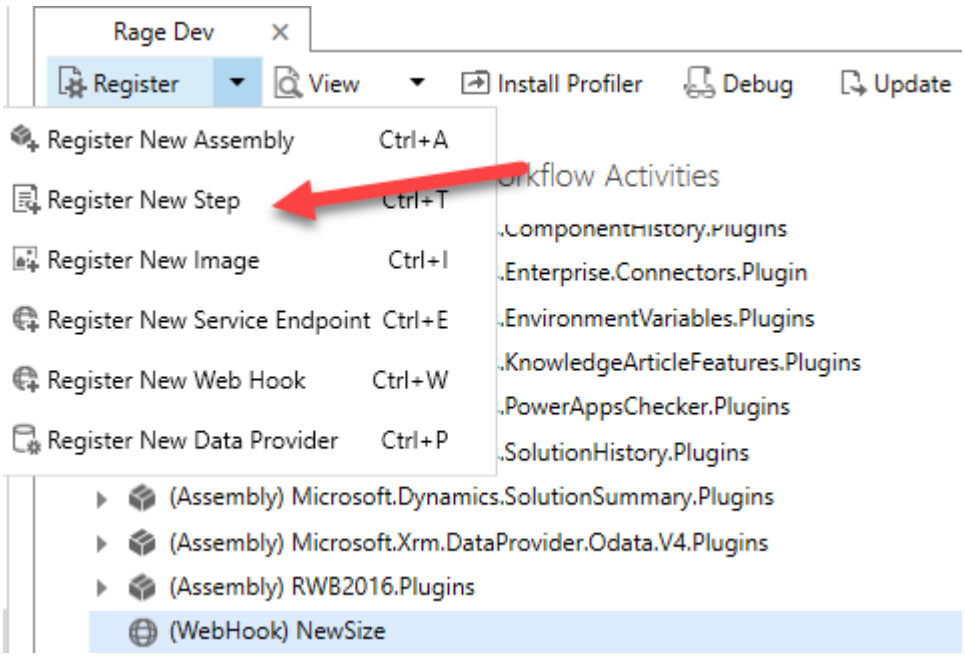
- Select **WebhookKey** for **Authentication**.
- Go back to the notepad and copy the key.

```
https://hr09092019.azurewebsites.net/api/HttpTrigger1?
code=ecVN3/JiBMDljpAn8W/xk1/oNose4NUFs14oexaBI9vasKcw3uksvg==
```

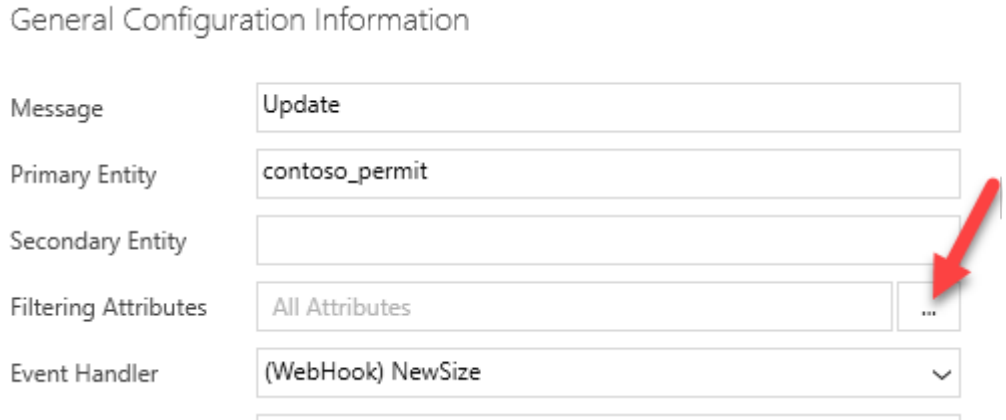
- Go back to the **Plugin Registration** tool, paste the key you copied in the **Value** Column and click **Save**.



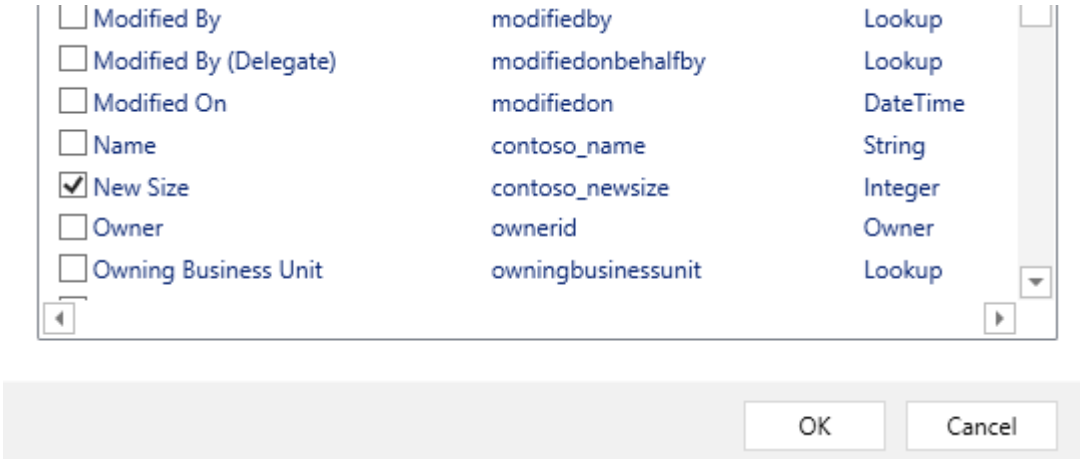
5. Register new step
   - Select the **Web Hook** you registered, click **Register** and select **Register New Step**.



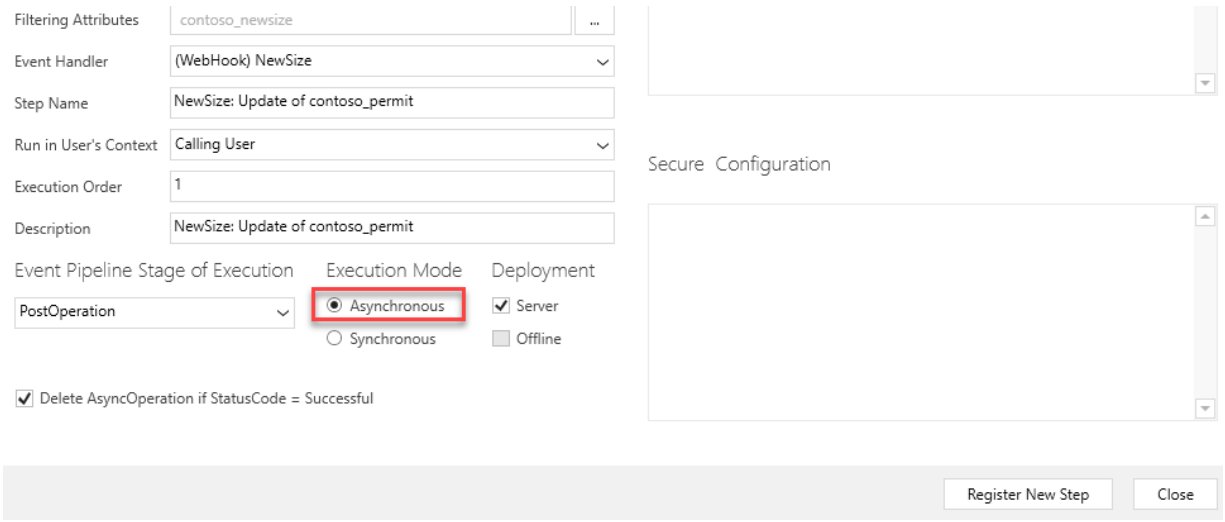
- Select **Update** for **Message**, **contoso_permit** for **Primary Table**, and click **Filtering Attributes.**



- Select only **New Size** and click **OK**.

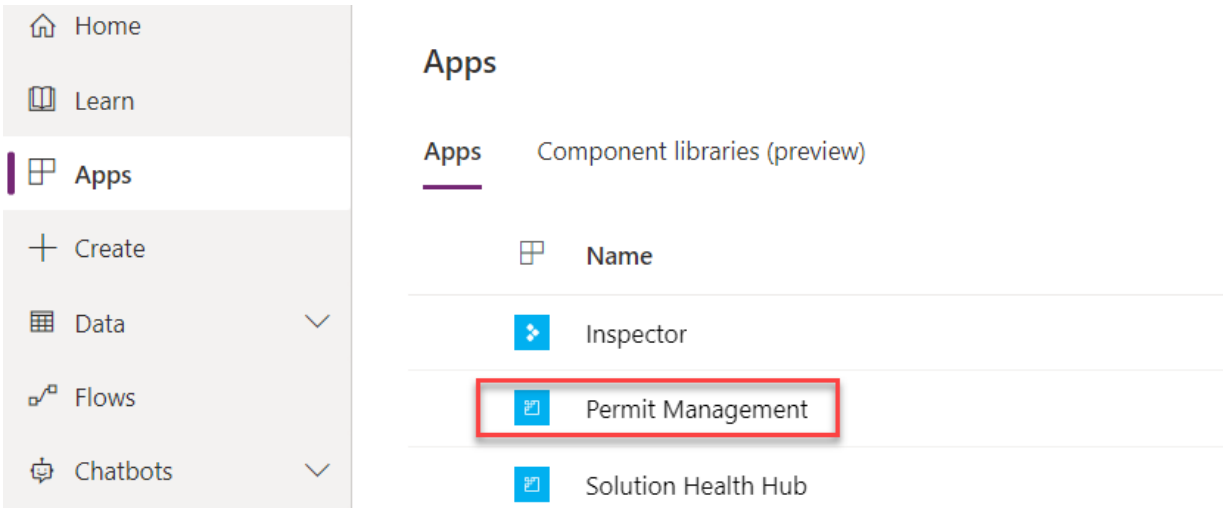


- Select **Asynchronous** for **Execution Mode** and click **Register New Step**.

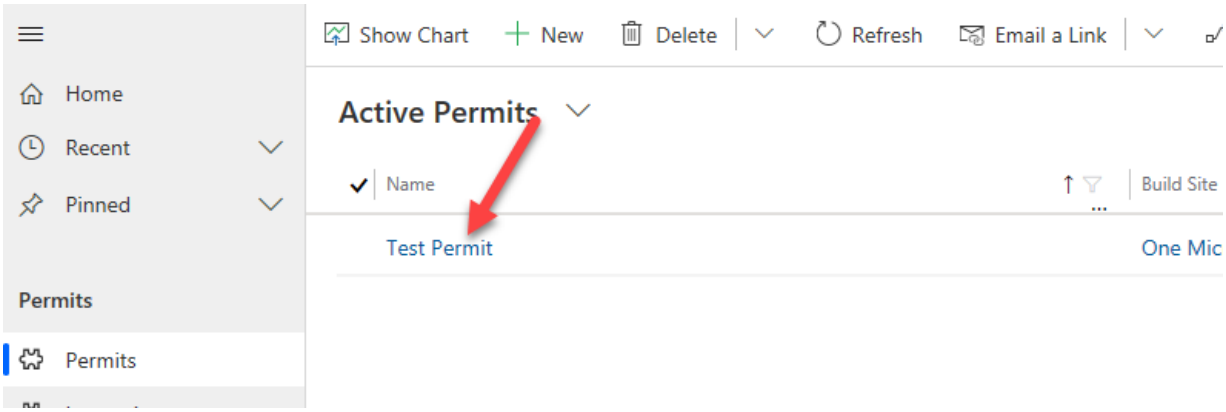


42.2 Task #2: Test the Web Hook

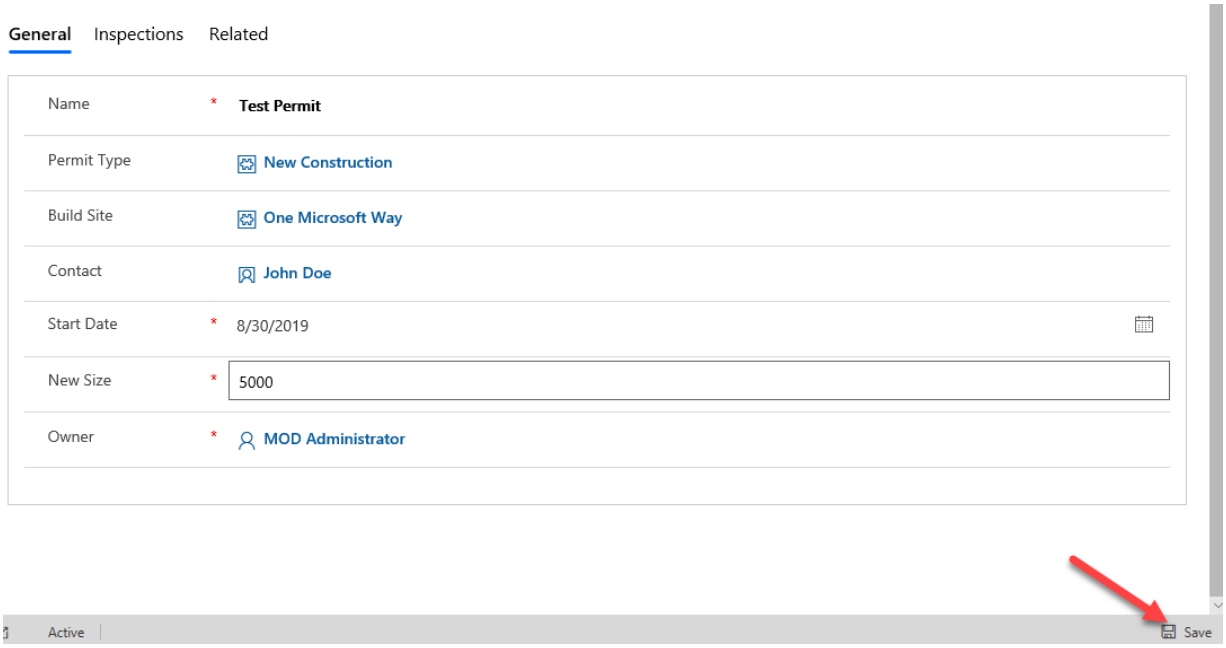
1. Start the Permit Management application
   - Sign in to Power Apps maker portal and make sure you have the **Dev** environment selected.
   - Select Apps and click to open the Permit Management application.



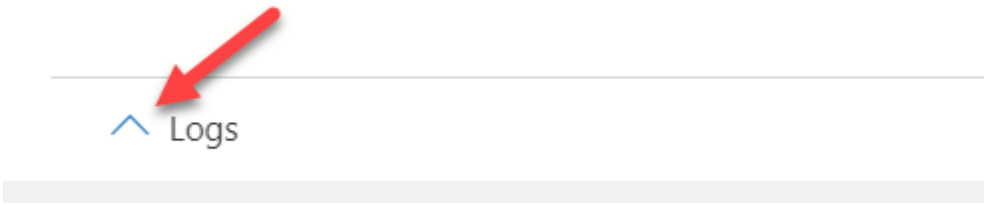
   - Select **Permits** and open one of the permit records. Create new if you don't have a Permit record.



   - Change the **New Size** to **5000** and **Save**.



2. Check Azure Output
   - Go back to your **Azure Function**.
   - Select **Code + Test**.
   - Show **Logs**.

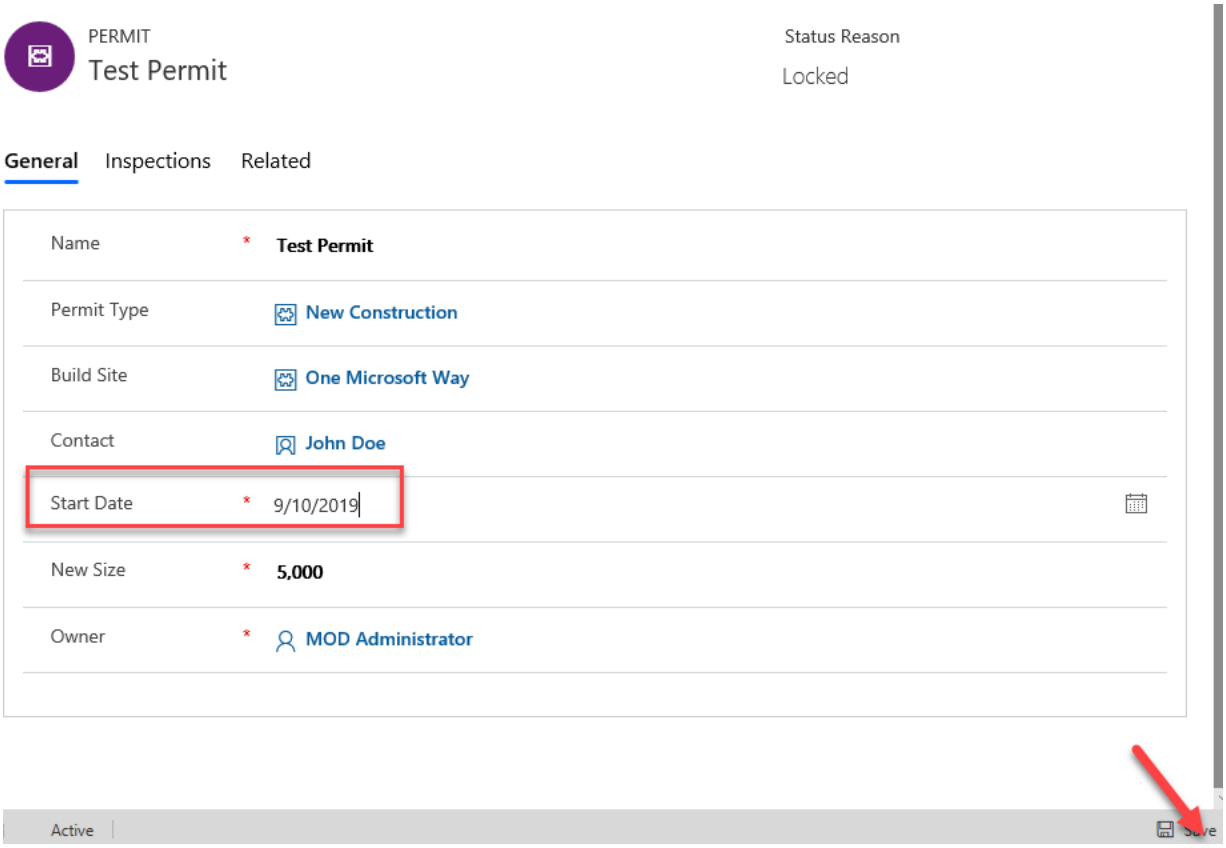


   - You should see logs like the image below. The Output is a serialized **RemoteExecutionContextobject** object

```
2020-08-14T16:18:30Z   [Information]   Executing 'Functions.HttpTrigger1' (Reason='This
19c9-8c6c-51d84facaa90)
2020-08-14T16:18:30Z   [Information]   C# HTTP trigger function processed a request.
2020-08-14T16:18:30Z   [Information]   {
  "BusinessUnitId": "0c543eea-65d9-ea11-a819-000d3a30f81d",
  "CorrelationId": "cb91b17d-b5a4-48a5-9b64-4b5aa7c6c9b2",
  "Depth": 1,
  "InitiatingUserAzureActiveDirectoryObjectId": "00000000-0000-0000-0000-000000000000",
  "InitiatingUserId": "0c1dd508-085a-4e63-8292-e42cb5e1563e",
  "InputParameters": [
    {
      "key": "Target",
      "value": {
        "__type": "Entity:http://schemas.microsoft.com/xrm/2011/Contracts",
        "Attributes": [
          {
            "key": "contoso_newsize",
            "value": 5000
          },
          {
            "key": "contoso_permitid",
            "value": "6536bd95-17dc-ea11-a816-00224803d22e"
```

Hint: If the log is not showing in the console (sometimes this happens), click **Monitor** on the left and check execution log. Select entry, details will be on the right (this could be delayed up to a few minutes).

6. Confirm the function executes only when the New Size value changes
    - Go back to the **Permit Management** application.
    - Change the **Start Date** to tomorrow's date and click **Save**.

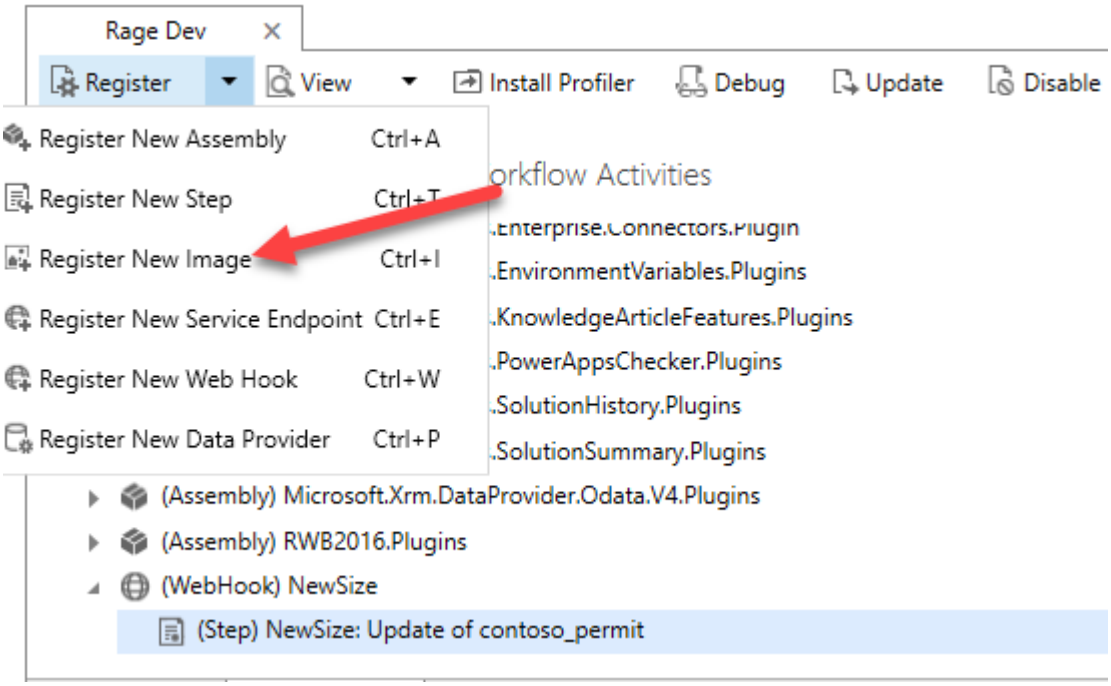


Go back to the Azure Function and make sure the function did not execute.

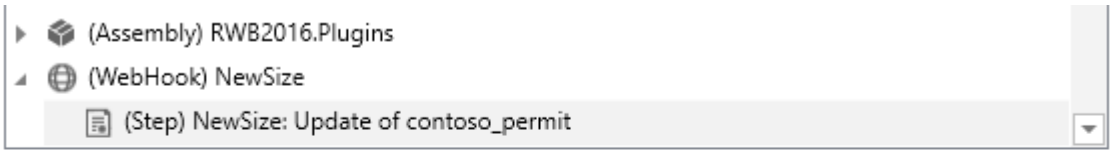
42.3 Task #3: Configure an Table image

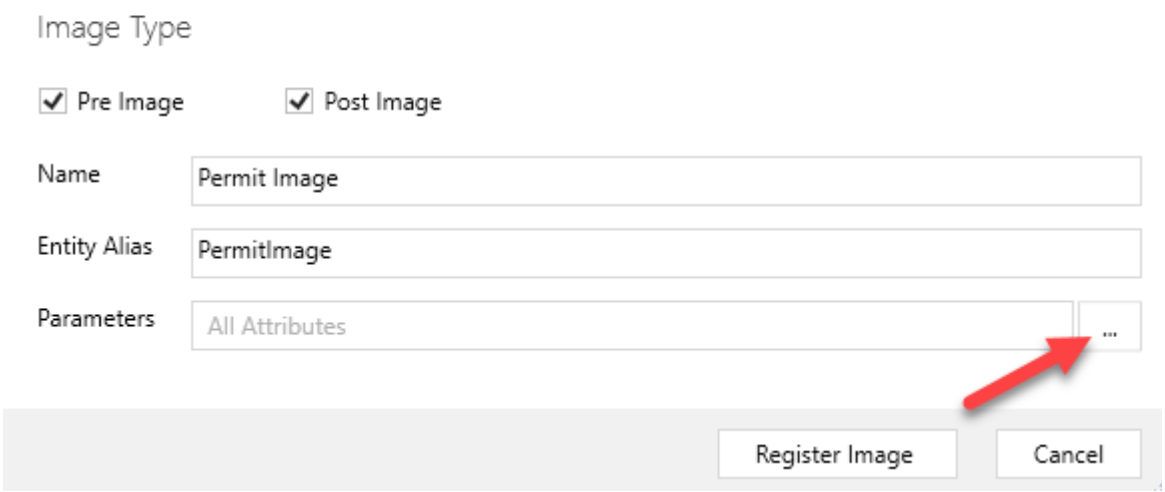
This step allows you to avoid unnecessarily querying Microsoft Dataverse and make a request only when you need information from the primary Table. It can also be used to get the prior value of a Column before an update operation.

1. Register New Image
    - Go back to the **Plugin Registration** tool.
    - Select the **NewSize** step you created, click **Register** and select **Register New Image**.

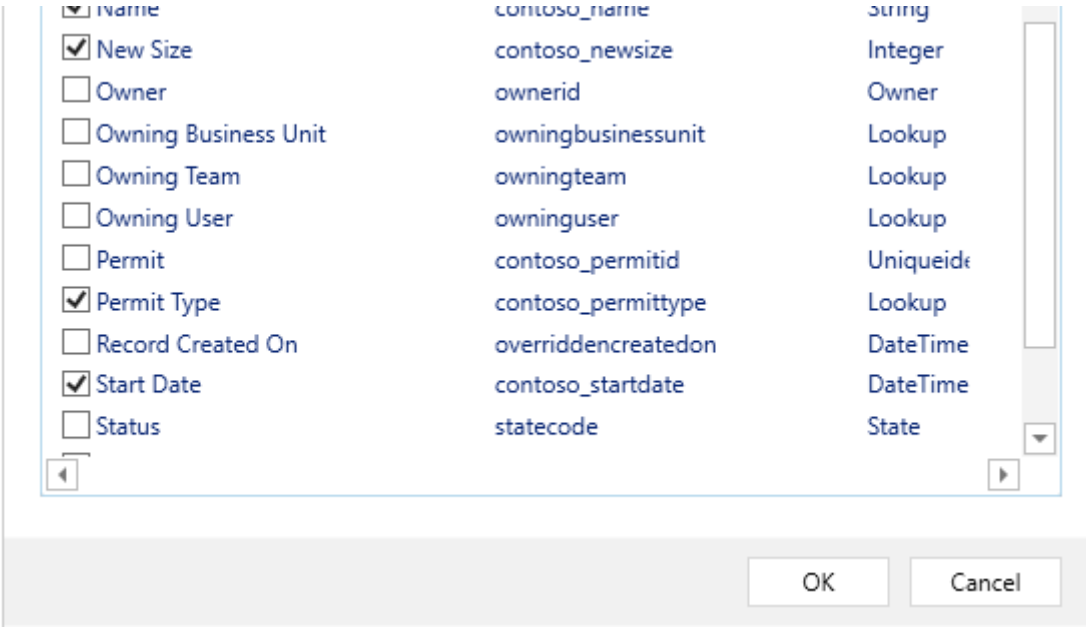


- Check both **Pre** and **Post** images checkboxes.
- Enter **Permit Image** for **Name**, **PermitImage** for **Table Alias**, and click on the **Parameters** button.

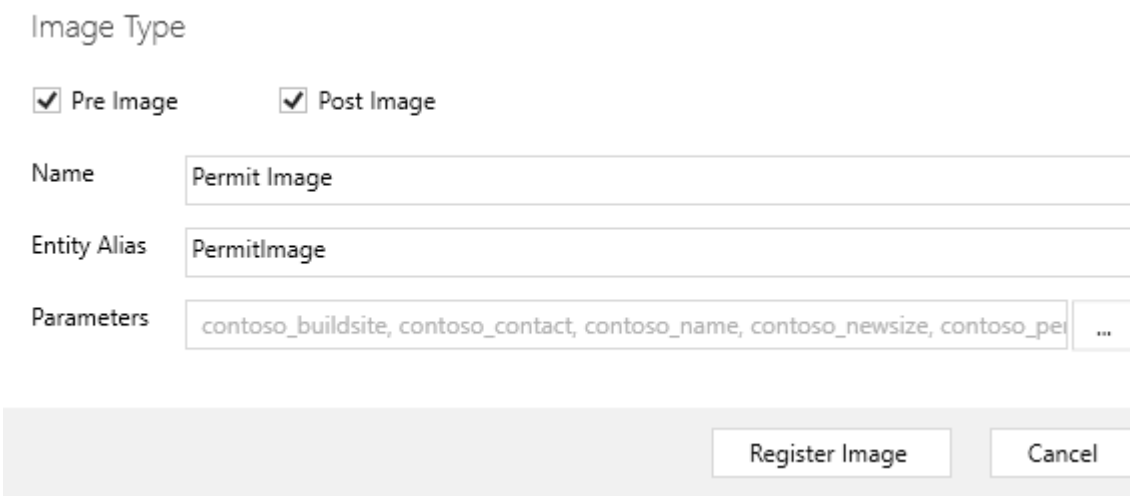




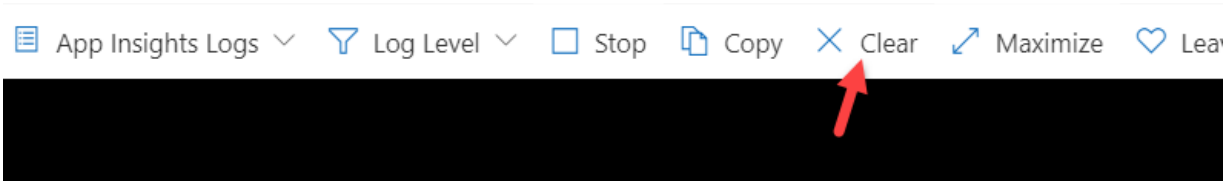
- Select **Build Site**, **Contact**, **Name**, **New Size**, **Permit Type**, and **Start Date**, and then click **OK**.



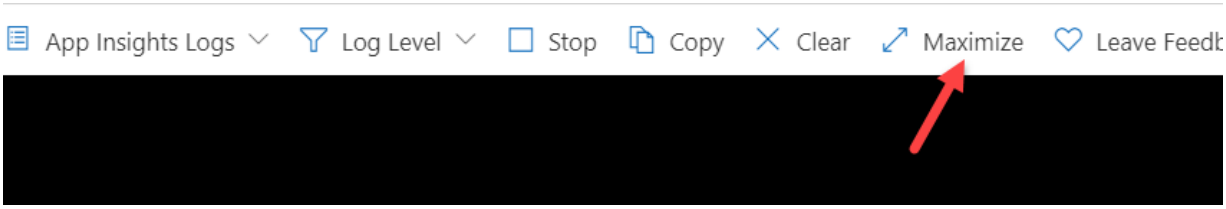
- Click **Register Image**.



2. Clear Azure log
   - Go back to the **Azure Function.**
   - Clear **Logs**.



3. Update Permit record
   - Go to the **Permit Management** application.
   - Select **Permits** and open one of the **Permit** records.
   - Change the New Size to **4000** and click **Save**
4. Check Azure logs
   - Go back to the **Azure Function**.
   - Maximize the log pane.



- The logs should now show both **Pre** and **Post** Table images. In this case you should see the old value **5000** in **Pre** image and the new value **4000** in the **Post** image

42.4 Note: Technically, we have the data in the target object already. However, if there are plugins modifying the data, PostImage will contain the copy as recorded in Microsoft Dataverse while Target contains the data as submitted on Save. In addition to that, preimage contains data before the save operation took place.

42.5 lab: title: 'Lab 09: Custom Connector'

[!NOTE] Effective November 2020:

- Common Data Service has been renamed to Microsoft Dataverse. Learn more
- Some terminology in Microsoft Dataverse has been updated. For example, *Table* is now *table* and *Column* is now *column*. Learn more

This content will be updated soon to reflect the latest terminology.

42.6 Lab 09 – Custom Connector

43 Scenario

A regional building department issues and tracks permits for new buildings and updates for remodeling of existing buildings. Throughout this course you will build applications and automation to enable the regional building department to manage the permitting process. This will be an end-to-end solution which will help you understand the overall process flow.

In this lab you will build a custom connector that can be used from Power Apps and Power Automate. Custom connectors describe existing APIs and allow them to be used easily. In this lab, you will build an API that has common calculations used by inspectors so that they can be used by applications. After building the API, you will create a custom connector definition to make it available to Power Apps and Power Automate.

The connector could have multiple actions defined on it. However, for our lab we will define a single action **Get Required CPM** – where CPM stands for Cubic Per Minute. In some regions this would be Cubic Feet Per Minute, and in others it could be Cubic Meters Per Minute. The action we are building will take the dimensions of a room and the number of air exchanges required by policy. The action logic will calculate the required CPM for that configuration. If you want, you can add additional actions to support other inspection type scenarios to the API and the custom connector.

After building the API and the custom connector you will modify the inspector app to add the user experience to use the connector. You will use the same connector and invoke an action from Power Automate.

44 High-level lab steps

As part of configuring the custom connector, you will complete the following

- Create an Azure function that implements the CPM API
- Create a custom connector in a solution
- Configure the security and actions on the custom connector
- Test the custom connector
- Configure a Power Apps canvas app to use the connector
- Configure a Power Automate to use the connector

44.1 Things to consider before you begin

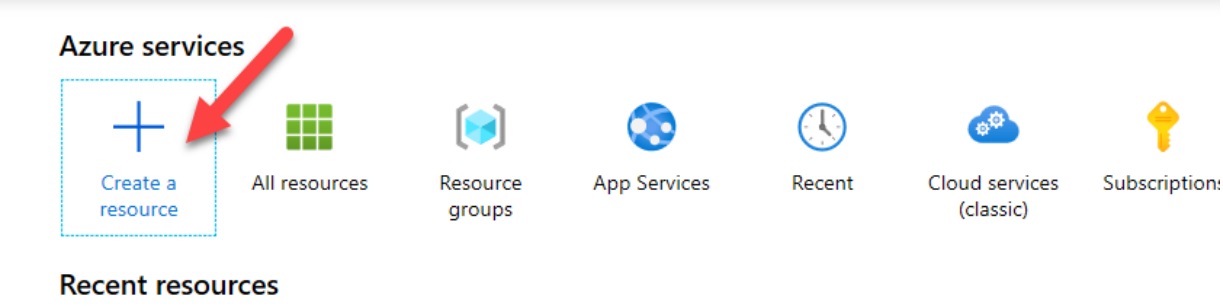
- Is there a standard approved connector already available that can be used?
- What security will we use in our connector?
- What are possible triggers and actions of the connector?
- Are there any API rate limits that could potentially affect the connector?

45 Exercise #1: Create the Azure Function

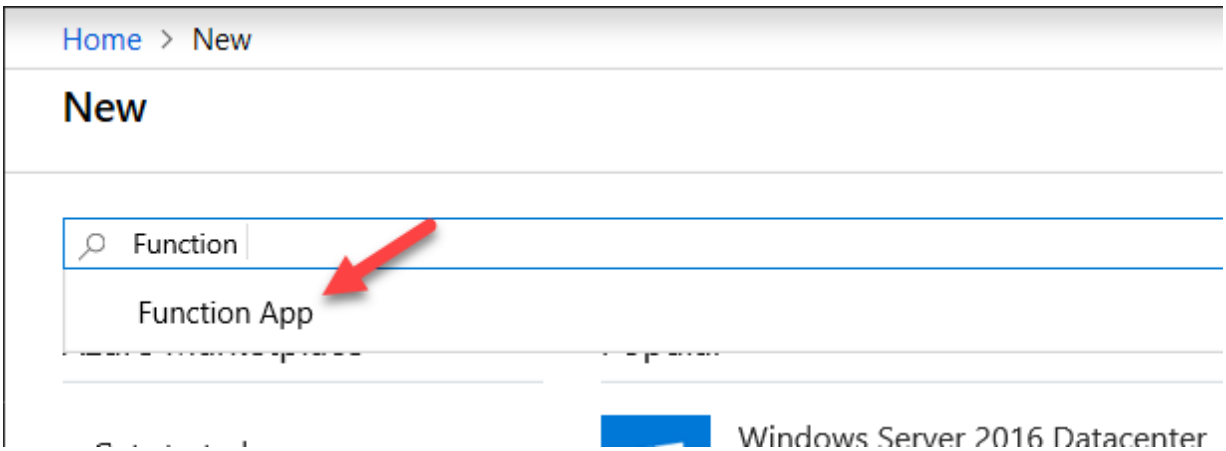
Objective: In this exercise, you will create an Azure function app and the function that will calculate the CPM.

45.1 Task #1: Create CPM Calculation Function

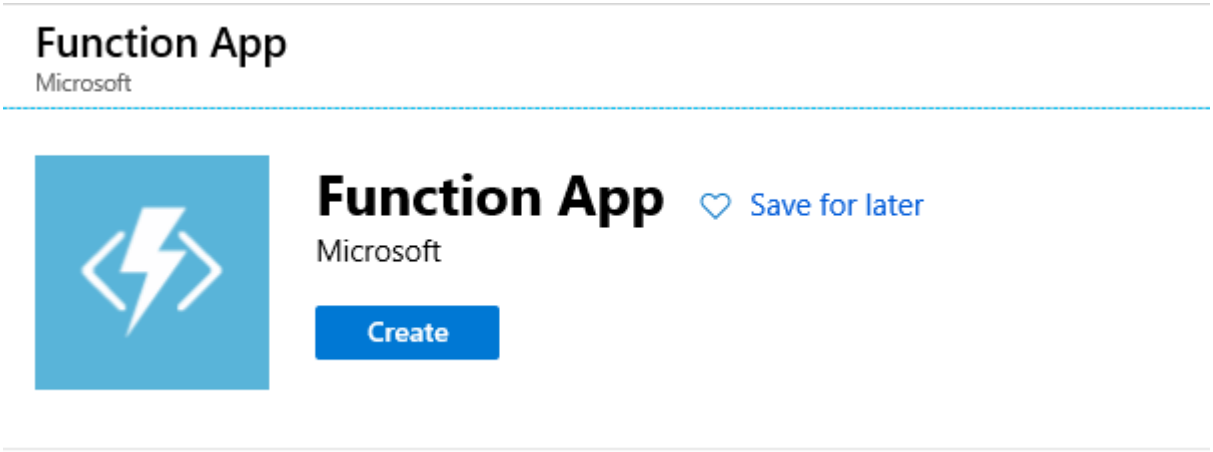
1. Create the function app
   - Sign in to Azure portal
   - Click **Create a Resource**.



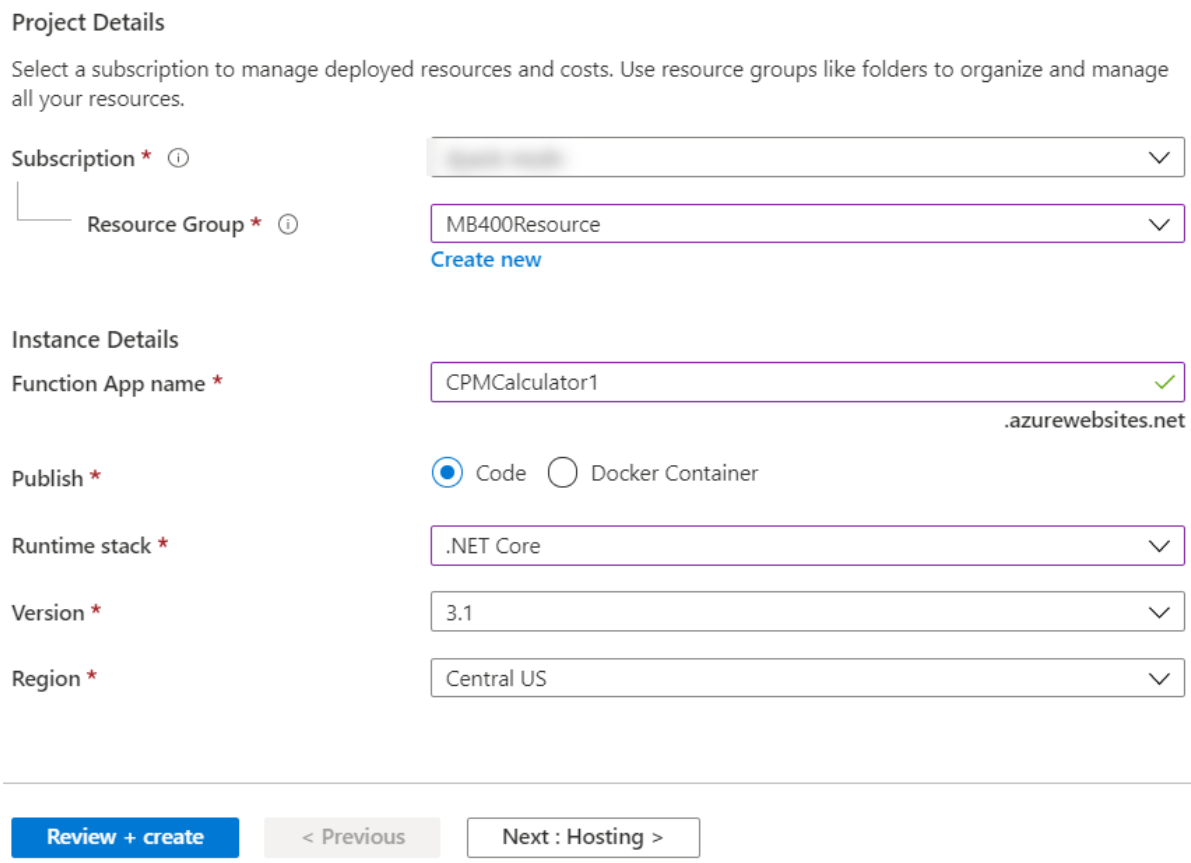
   - Search for **Function** and select **Function App**.



- Click **Create**.

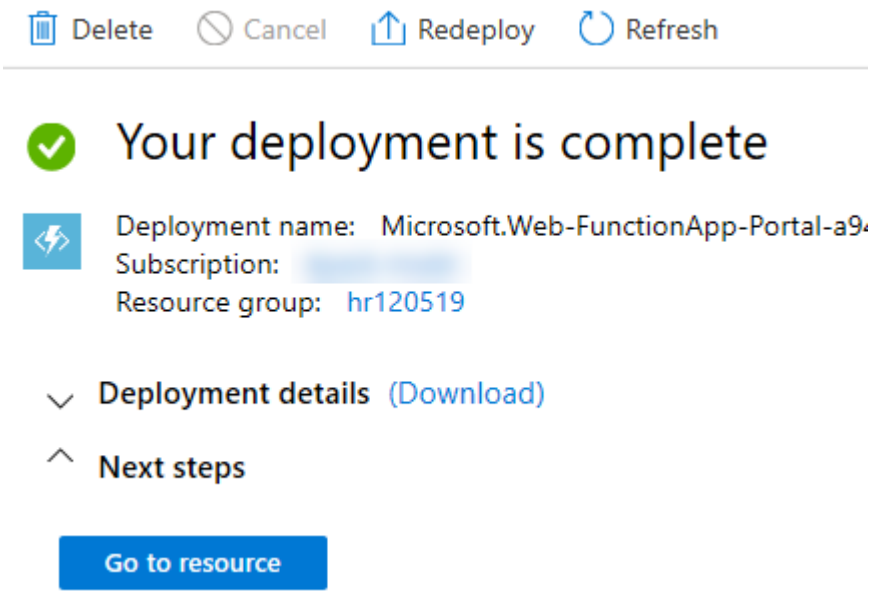


- Enter **CPMCalculator** for **App Name**. Provide a unique name, if CPMCalculator is not available.
- Create **New Resource Group**. You may select an existing Resource Group if you already created one for this course.
- Select **.NET Core** for **Runtime Stack**, select your **Region** and click **Review + Create**.

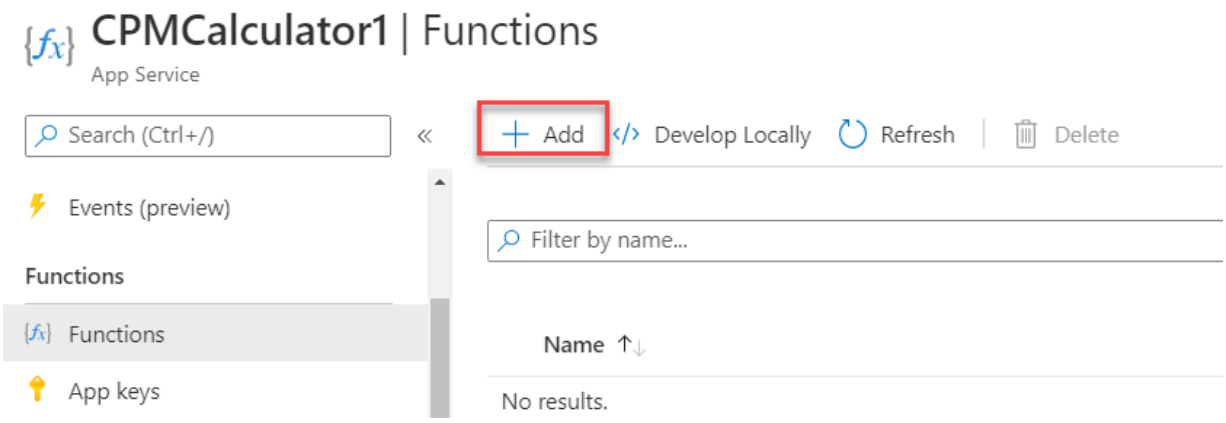


- Click **Create**. Wait for the function app to be created

2. Create function
   - Click **Go to Resource**.

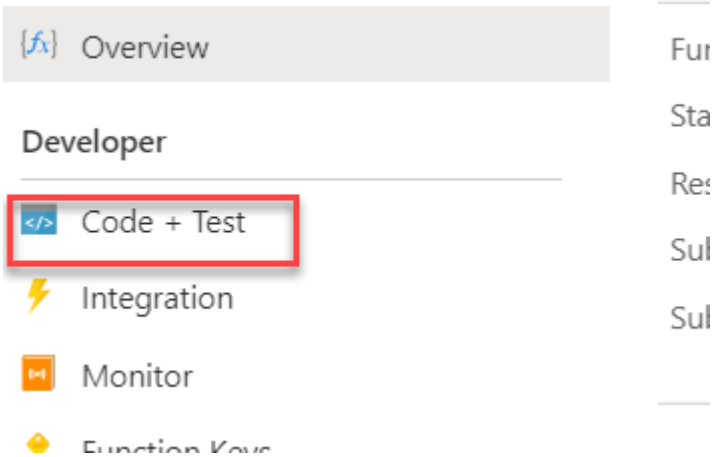


- Select **Functions** and click **+ Add**.



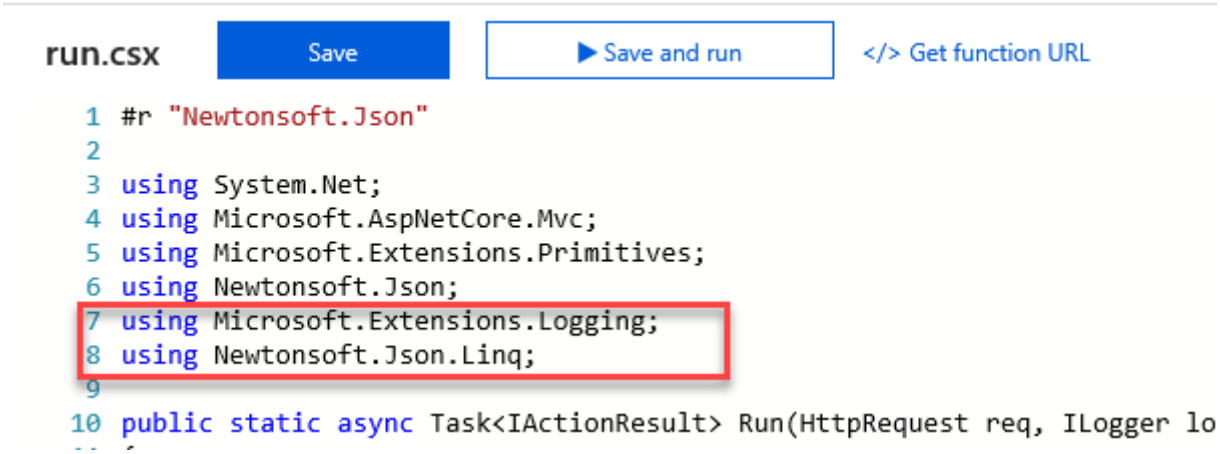
- Select **HTTP trigger**.
- Click **Create Function.**

3. Add the **Using Statements** and **CRMCalcRequest** class to the function.
   - Select **Code + Test**.



   - Add the Using Statements below to the function.

```
using Microsoft.Extensions.Logging;
using Newtonsoft.Json.Linq;
```



   - Add the public class below to the function. This will describe the request that will be sent from the applications using the API.

```
public class CPMCalcRequest
{
    public int Width=0;
    public int Height=0;
    public int Length=0;
```

```
        public int AirChanges=0;
    }
17
18      return name != null
19          ? (ActionResult)new OkObjectResult($"Hello, {name}")
20          : new BadRequestObjectResult("Please pass a name on the query string or in the r
21  }
22
23  public class CPMCalcRequest
24  {
25
26      public int Width=0;
27      public int Height=0;
28      public int Length=0;
29      public int AirChanges=0;
30
31  }
32
33
```

4. Clean up the Run method
   - Go to the **Run** method.
   - Remove everything but the log line from the **Run** method.

```
public static async Task<IActionResult> Run(HttpRequest req, ILogger log)
{
    log.LogInformation("C# HTTP trigger function processed a request.");

}
```

5. Get the Request body and deserialize it as **CRMCalcRequest**
   - Get the request **Body** from the request argument. Add the code below inside the **Run** method.

     ```
     string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
     ```

   - Deserialize the request body. Add the code below inside the **Run** method.

     ```
     CPMCalcRequest calcReq = JsonConvert.DeserializeObject<CPMCalcRequest>(requestBody);
     ```

```
8   public static async Task<IActionResult> Run(HttpRequest req, ILogger log)
9   {
10      log.LogInformation("C# HTTP trigger function processed a request.");
11
12      string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
13      CPMCalcRequest calcReq = JsonConvert.DeserializeObject<CPMCalcRequest>(requestBody);
14  }
15
```

6. Calculate the CPM and return it form the Run method
   - Calculate the **CPM** and log the calculated value. Add the code below inside the **Run** method.

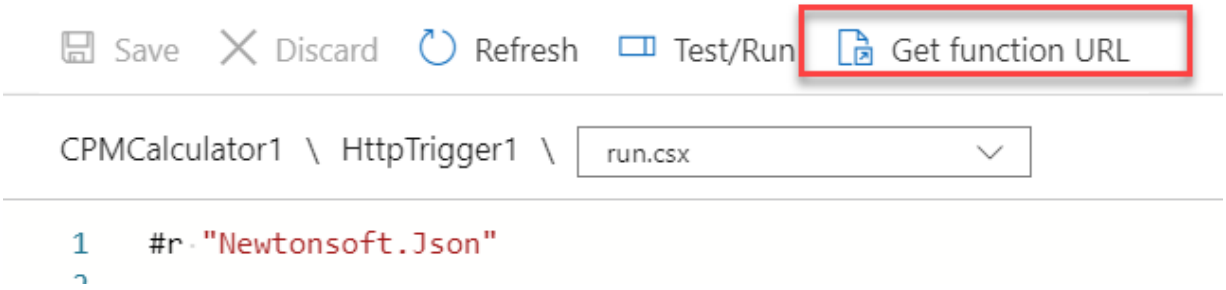
     ```
     var cpm = ((calcReq.Width*calcReq.Height*calcReq.Length) * calcReq.AirChanges) / 60;
     log.LogInformation("CPM " + cpm);
     ```

   - Return the calculated **CPM**. Add the code below inside the Run method.

     ```
     return (ActionResult)new OkObjectResult(new{
         CPM = cpm
     });
     ```

```
/
public static async Task<IActionResult> Run(HttpRequest req, ILogger log)
{
    log.LogInformation("C# HTTP trigger function processed a request.");

    string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
    CPMCalcRequest calcReq = JsonConvert.DeserializeObject<CPMCalcRequest>(requestBody);

    var cpm = ((calcReq.Width*calcReq.Height*calcReq.Length) * calcReq.AirChanges) / 60;
    log.LogInformation("CPM " + cpm);

    return (ActionResult)new OkObjectResult(new{
        CPM = cpm
    });
}
```

7. Click **Save** to save your changes.
8. Copy the Function URL.
   - Click **Get Function URL**.



   - Click **Copy**.

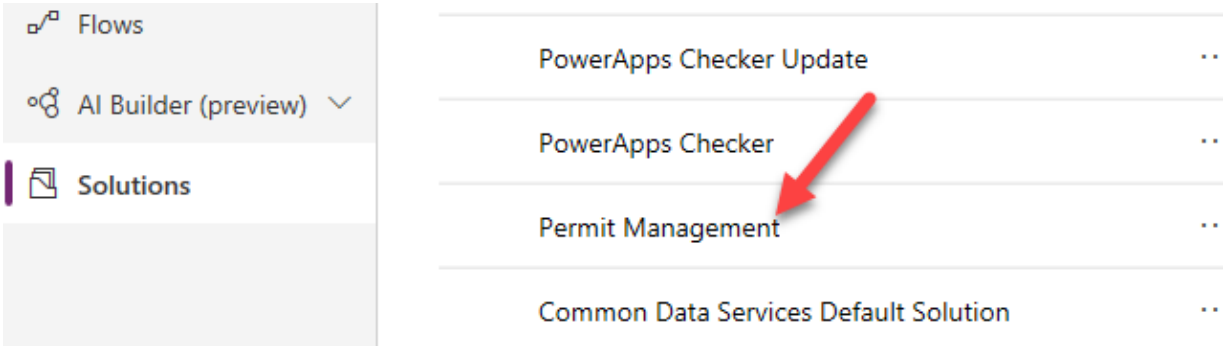
   - Keep the URL you copied on a notepad. You will need this URL while creating the custom connector.

46 Exercise #2: Create the Custom Connector

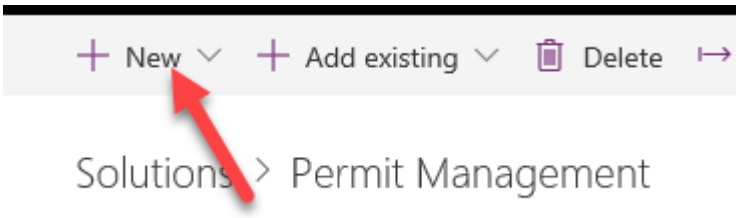
Objective: In this exercise, you will create the Custom Connector. This same approach could be used to describe any existing API you create or that has been created by any 3rd parties and an existing connector for that service is unavailable.

46.1 Task #1: Create the Custom Connector

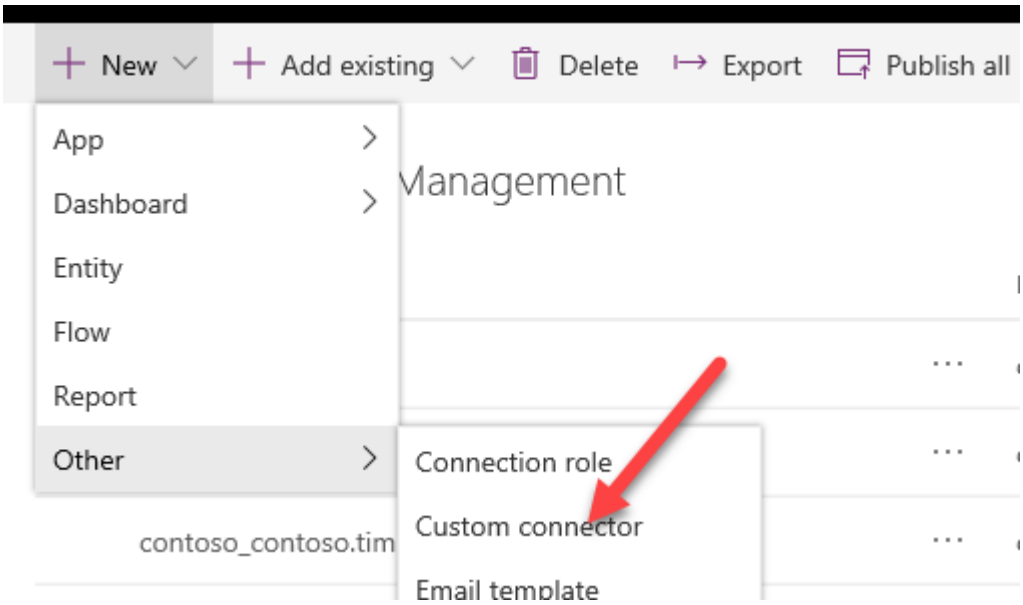
1. Open the Permit Management solution
   - Sign in to Power Apps maker portal and make sure you are in the **Dev** environment.
   - Select **Solutions**.
   - Click to open the **Permit Management** solution.



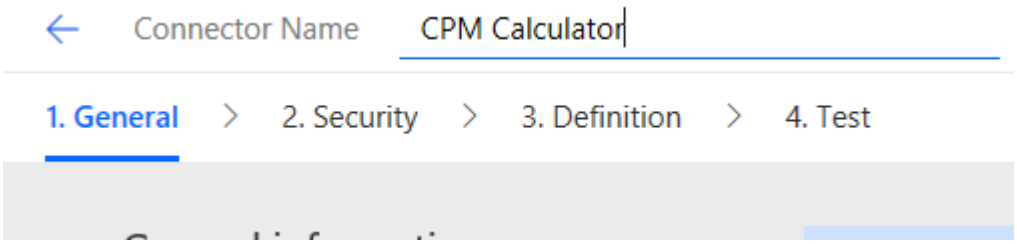
2. Create Custom Connector
   - Click + **New**.



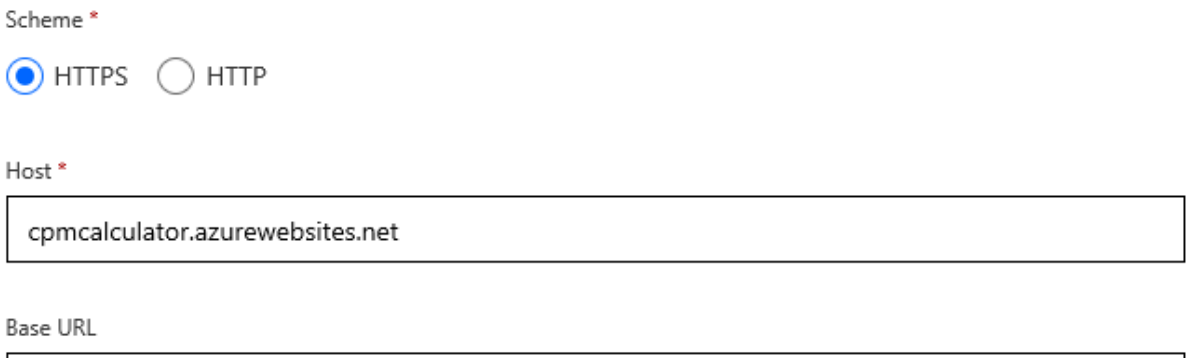
   - Select **Other| Custom Connector**.



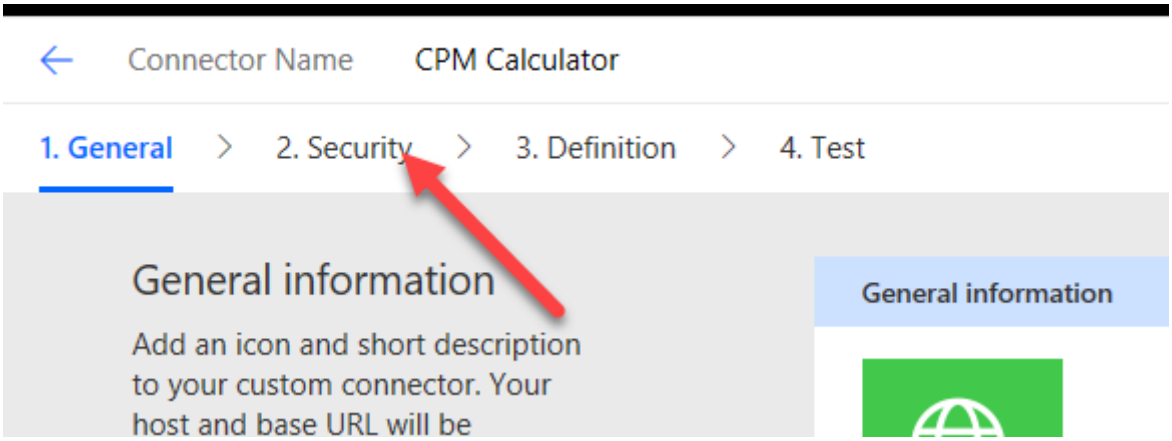
   - Change the **Connector Name** from **Untitled** to **CPM Calculator**.



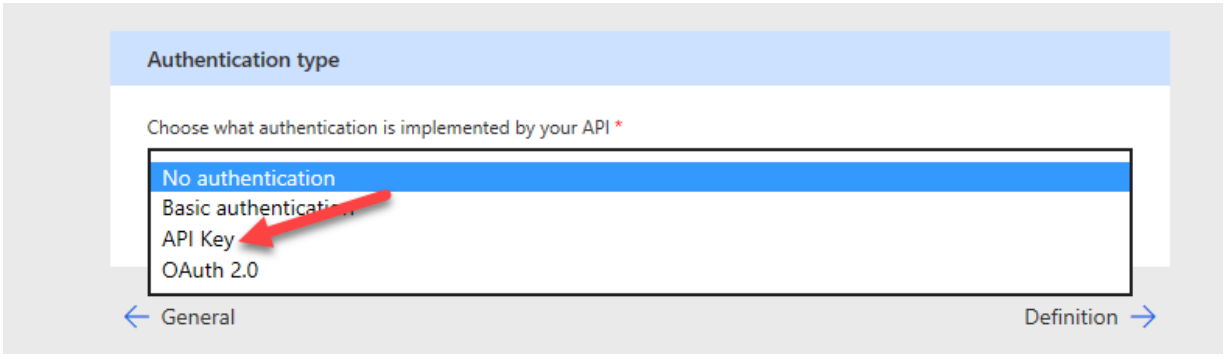
   - Locate the **Host** Column and paste the **Function URL** you copied in Exercise 1.
   - Remove https:// and everything after .net.



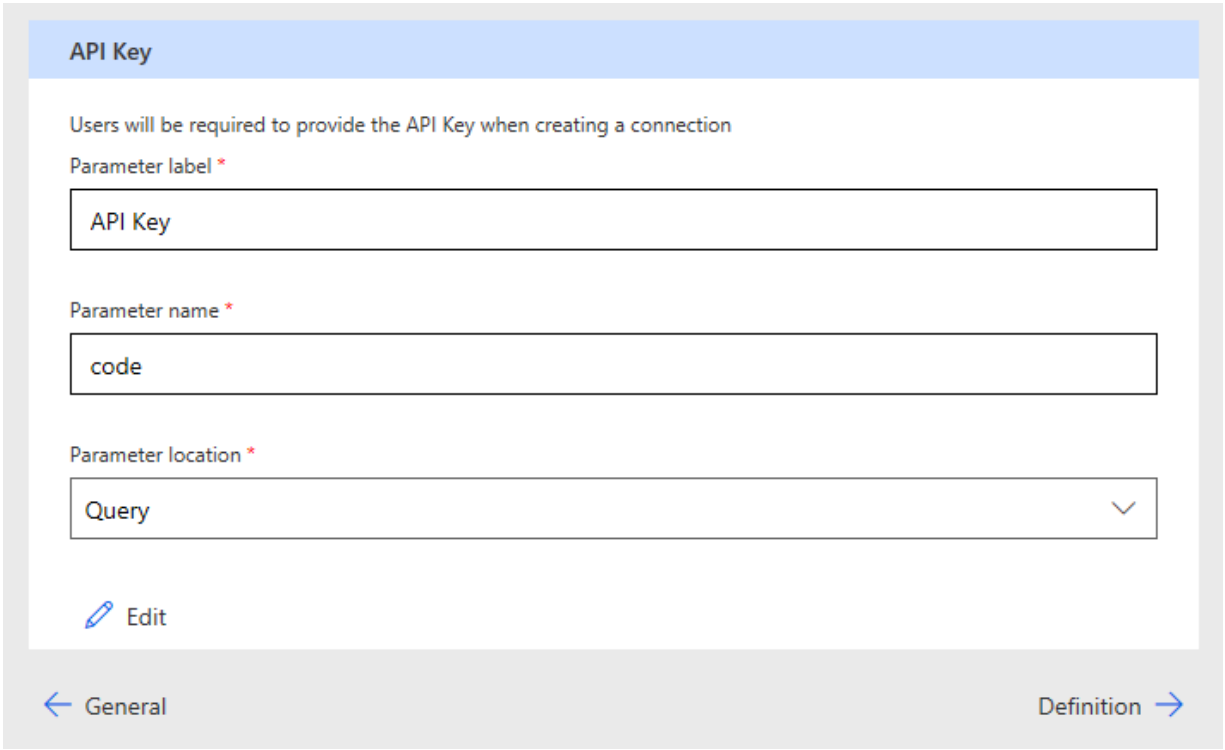
3. Select API key for security and create the connector
   - Advance to **Security**.



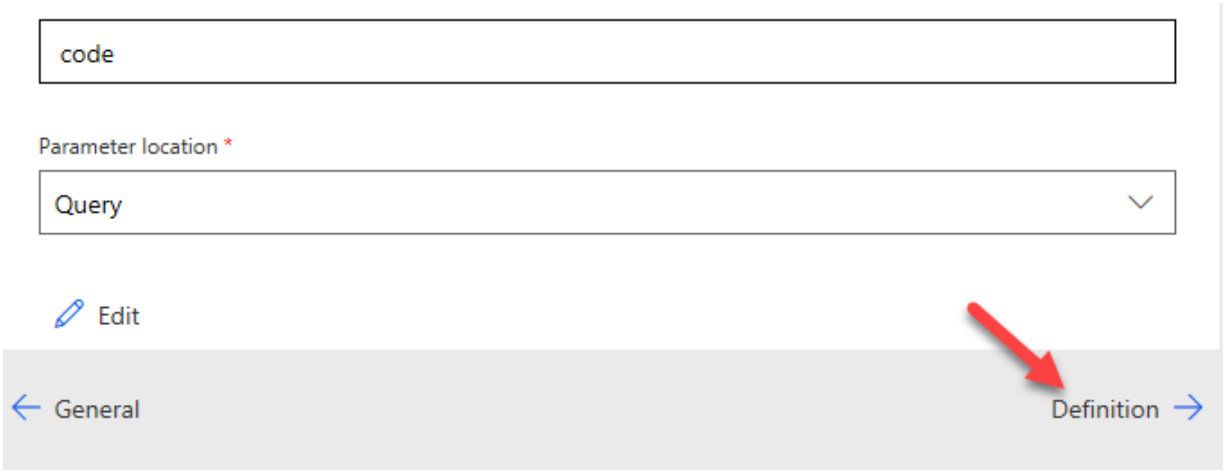
   - Select **API Key**.



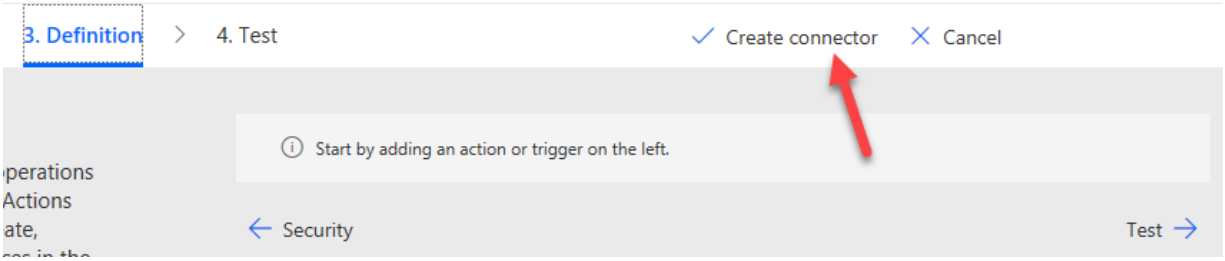
   - Enter **API Key** for **Parameter Label**, **code** for **Parameter Name**, and select **Query** for **Parameter Location**.



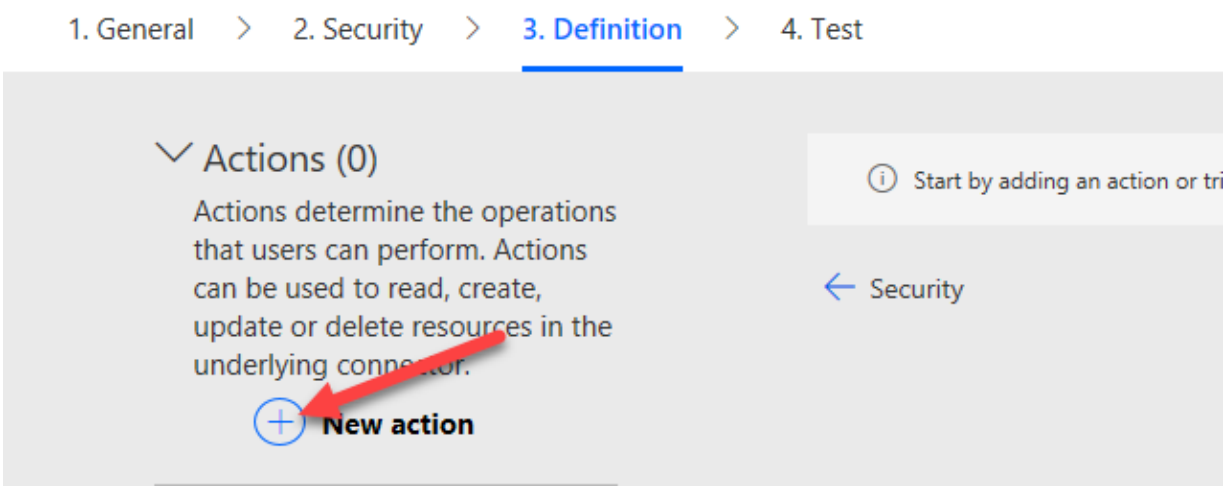
4. Create Connector
   - Advance to **Definition**.



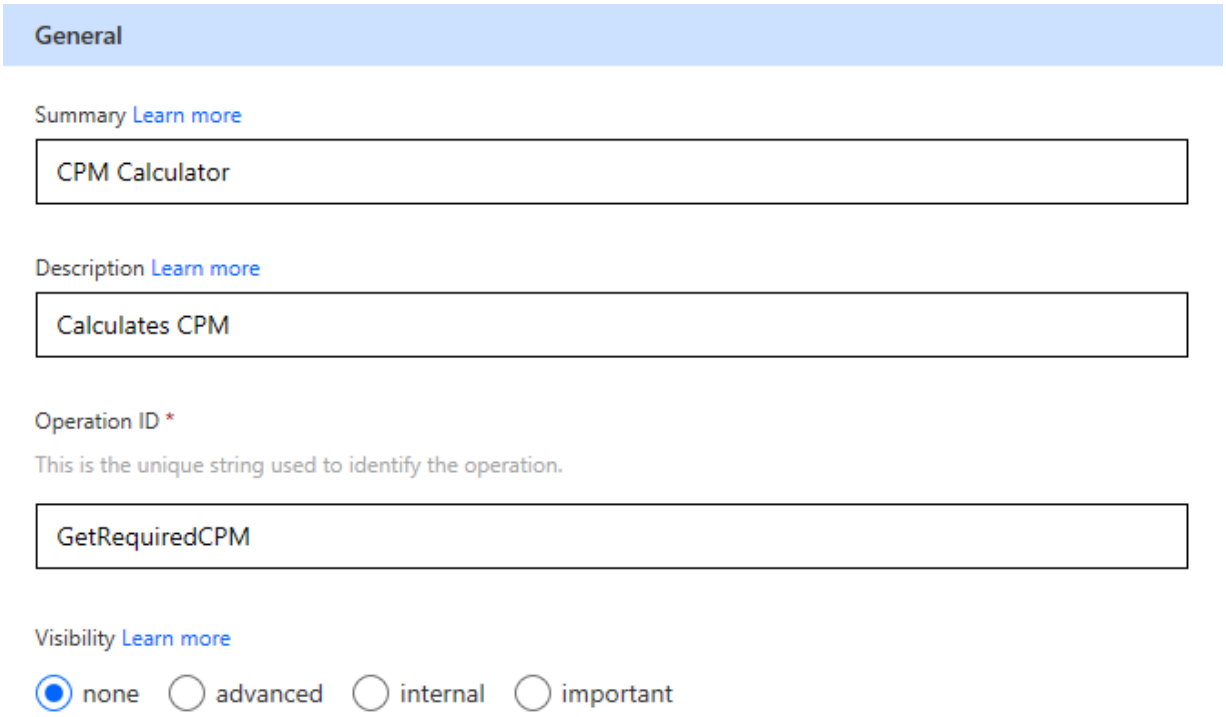
   - Click **Create Connector** and wait for the connector to be created.



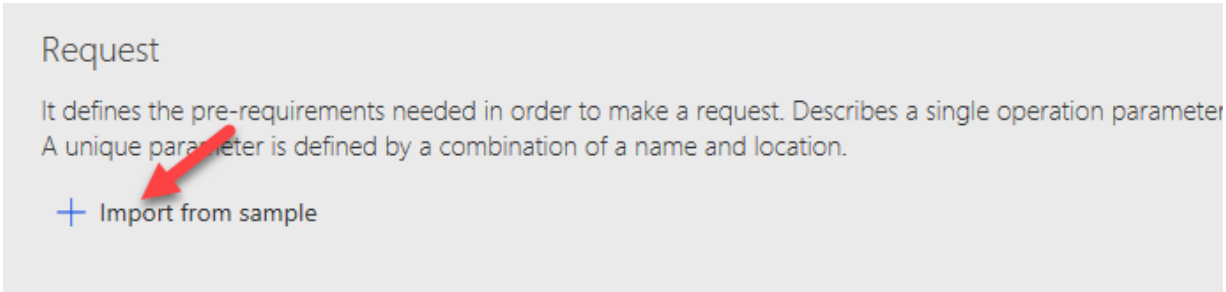
5. Create Action
   - Click **New Action**. The action describes each operation that the API has. These can be manually defined like we are doing here or can be imported from Open API or Postman collection files for larger APIs.



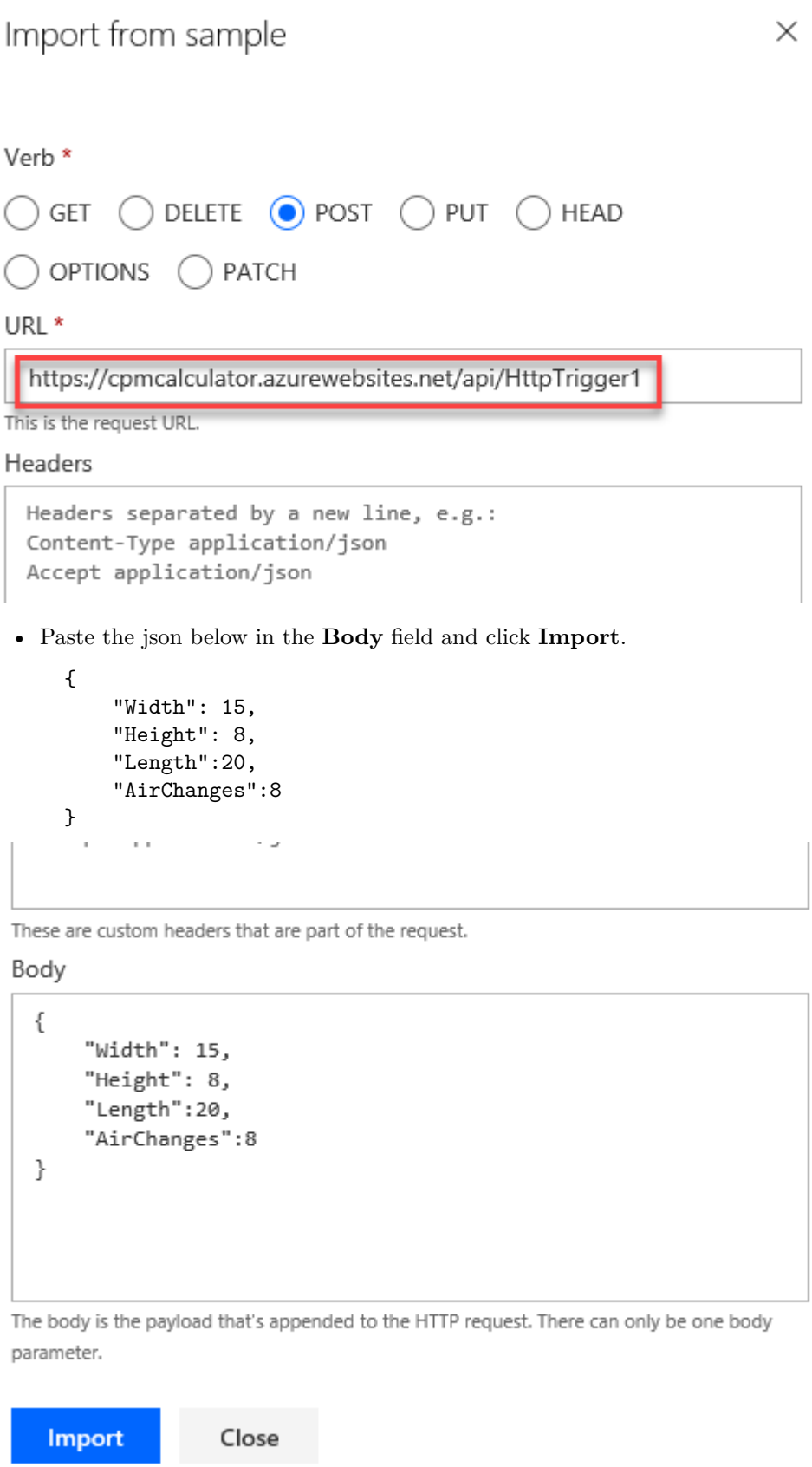
- Enter **CPM Calculator** for **Summary**, **Calculates CPM** for **Description**, and **GetRequiredCPM** for **Operation ID**.



6. Import request from sample
   - Click + **Import from Sample**.



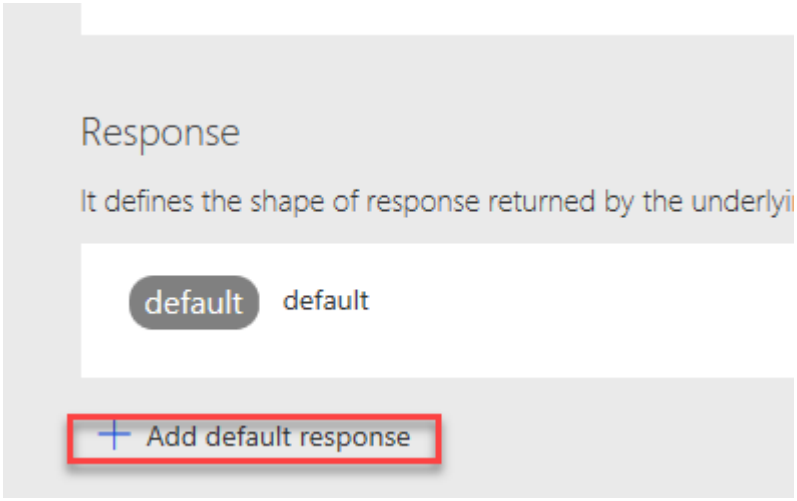
   - Select **Post** for **Verb**.
   - Paste the function **URL** from your notepad and remove everything after **HttpTrigger1**.



- Paste the json below in the **Body** field and click **Import**.

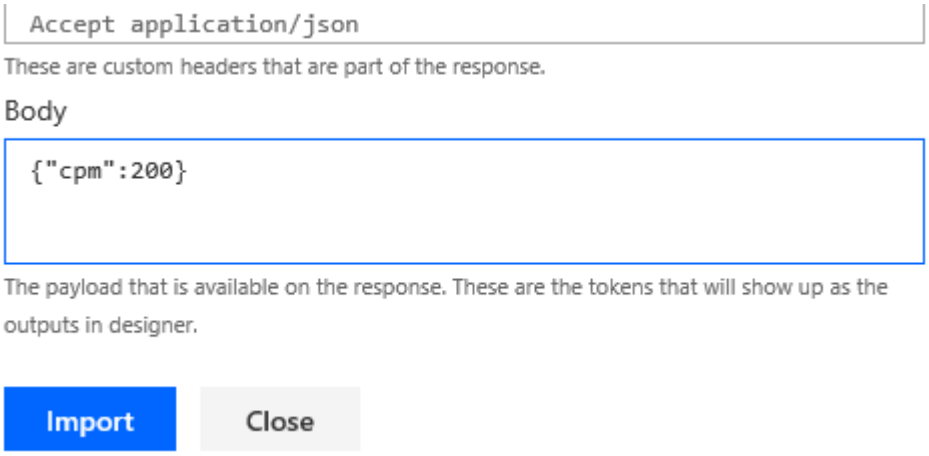
```
{
    "Width": 15,
    "Height": 8,
    "Length":20,
    "AirChanges":8
}
```

7. Add Default response
    - Scroll down to the **Response** section and click + **Add Default Response**.

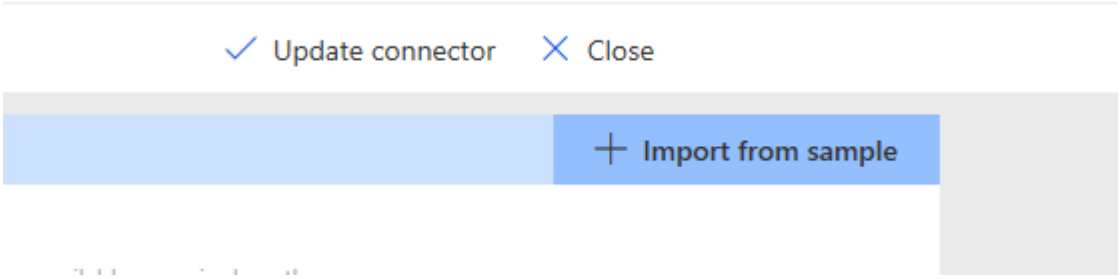


- Paste the json below in the **Body** and click **Import**.

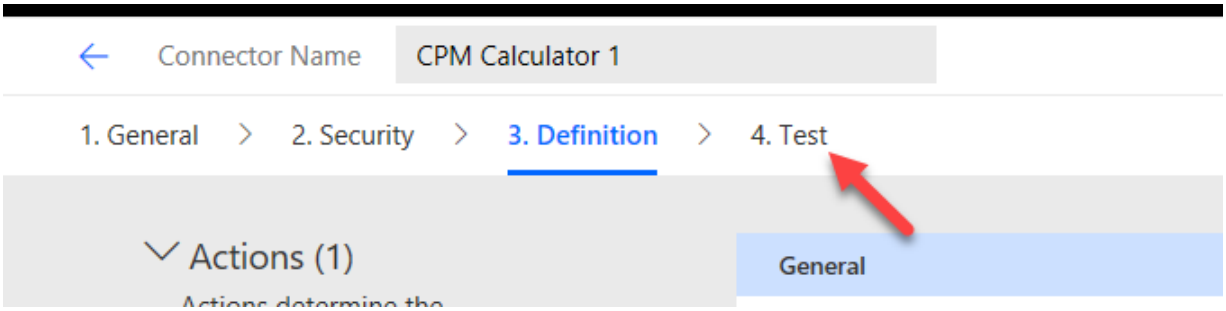
```
{"cpm":200}
```



- Click **Update Connector**.



8. Test the connector
   - Advance to Test.

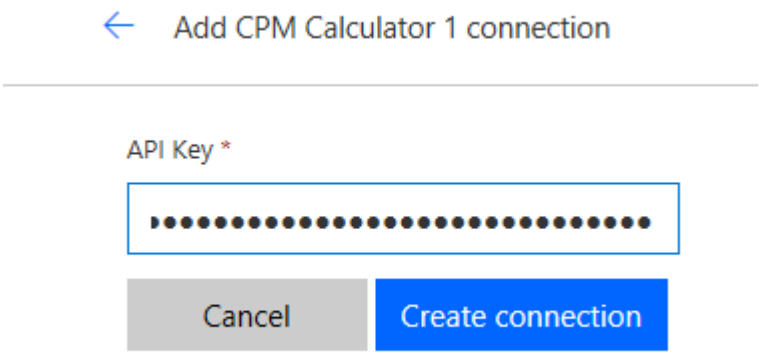


- Click **New Connection**. This will open a New window.

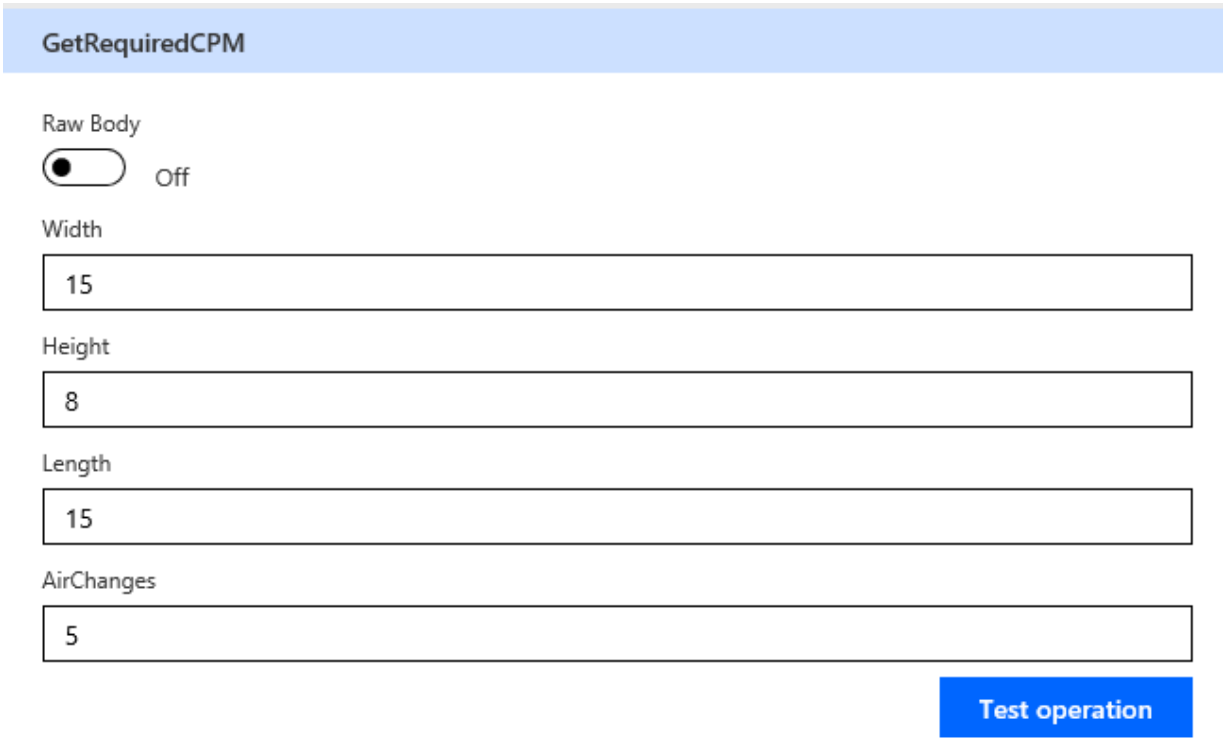
- Go to your notepad and copy only the value of the **code**.

```
https://cpmcalculator.azurewebsites.net/api/HttpTrigger1?
code=WlRANA62qDDoi6XuG/4D03HWJbQ3zCzTora3edXqr02lCEdKkpocaA==
```

- Go back to the connector, paste the value you copied, and click **Create Connection**.



- Refresh the connections and select newly created connection.
- Provide values for **Width**, **Height**, **Length**, and **AirChanges**.
- Click **Test Operation**.



- You should get a CPM value back.

- Close the window and go back to Solution window and click **Done.**
- Click **Publish all Customizations.**

47 Exercise #3 Test Connector

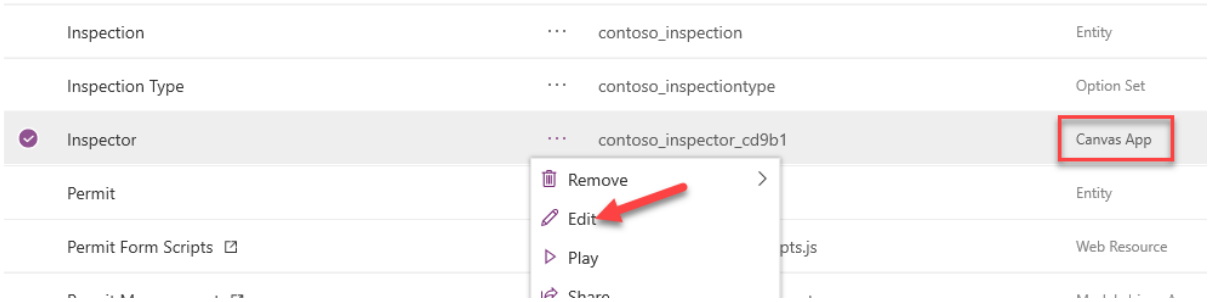
Objective: In this exercise, you will use the Custom Connector from a Power 9Apps canvas app and a Power Automate.

47.1 Task #1: Test on Canvas App

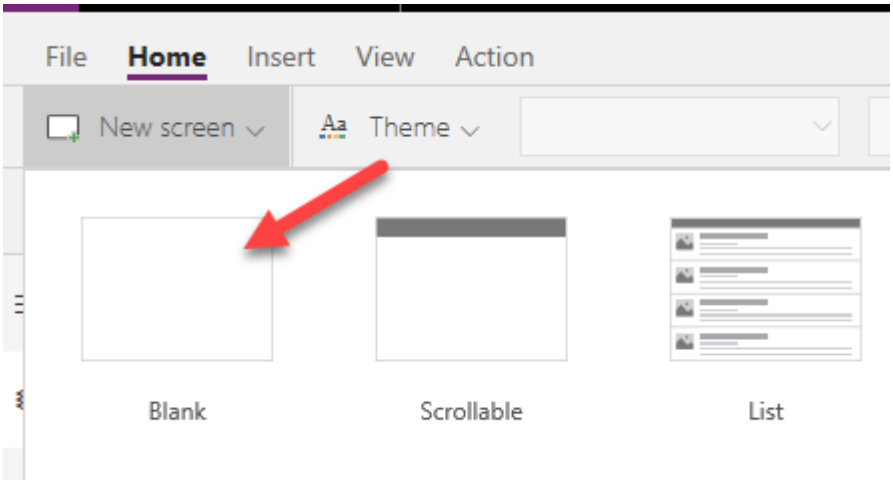
1. Open the Permit Management solution

- Sign in to Power Apps maker portal and make sure you are in the **Dev** environment.
- Select **Solutions.**
- Click to open the **Permit Management** solution.

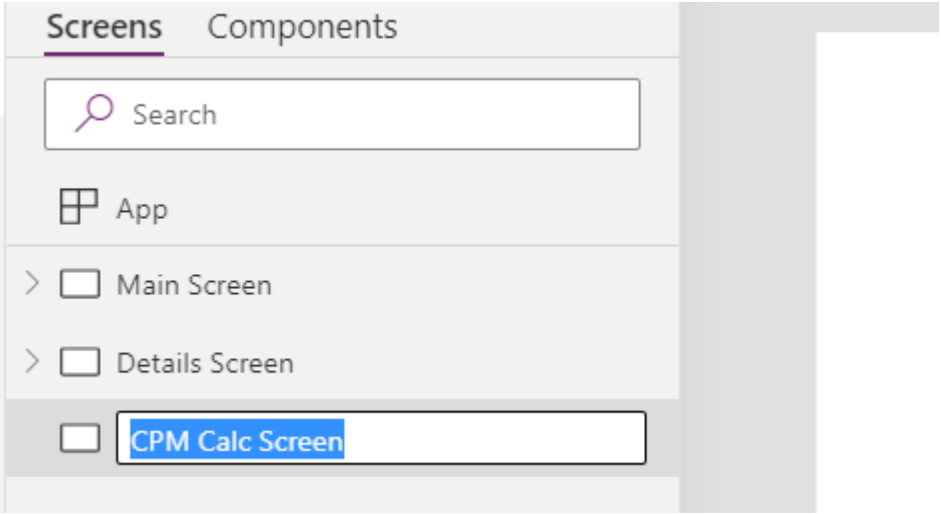
2. Open the Inspector canvas application in edit mode
   - Locate the **Inspector Canvas App**.
   - Click on the **... More Commands** button and select **Edit**. This will open the app in a New window.



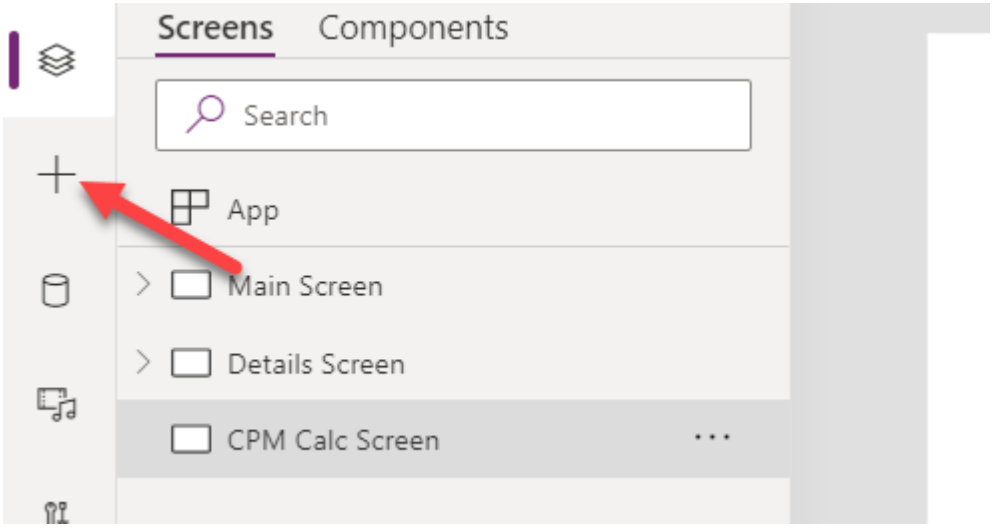
3. Add new screen to the application
   - Click **New Screen** and select **Blank**.



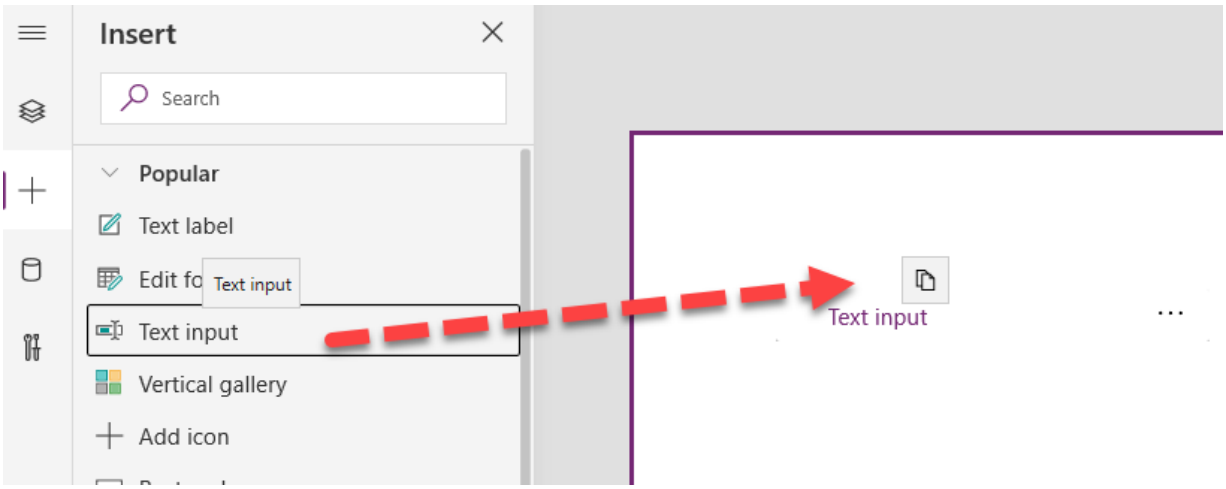
   - Rename the screen **CPM Calc Screen**



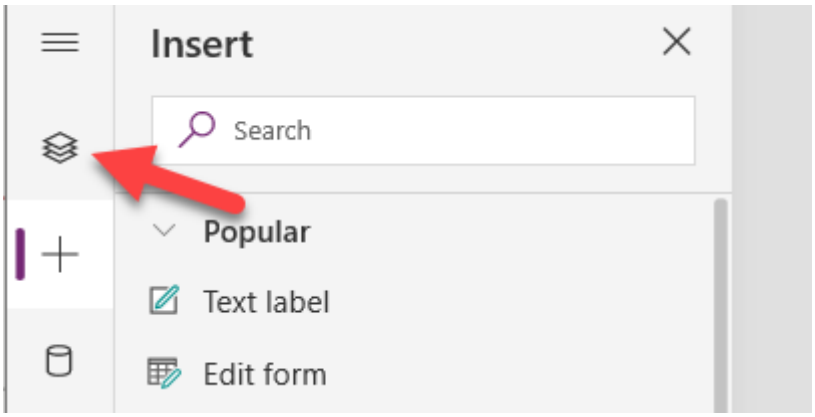
4. Add Input Text to the new screen
   - Select the **CPM Calc Screen**.
   - Click **Insert**.



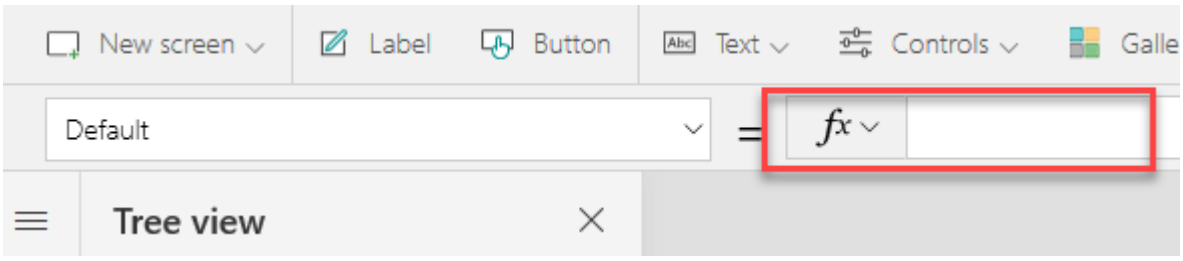
- Drag and drop **Text Input** to the screen.



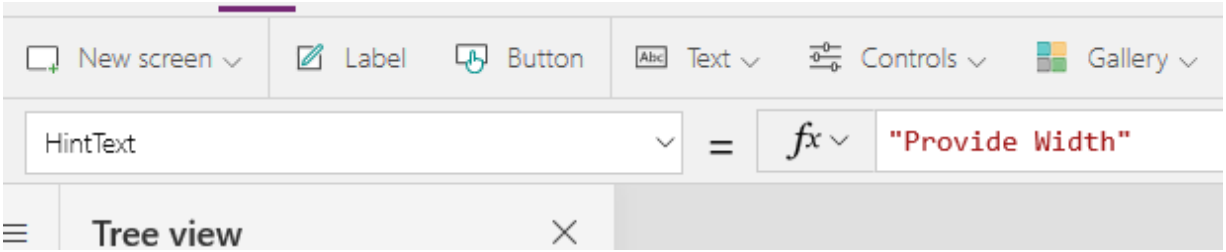
- Select the **Tree View**.



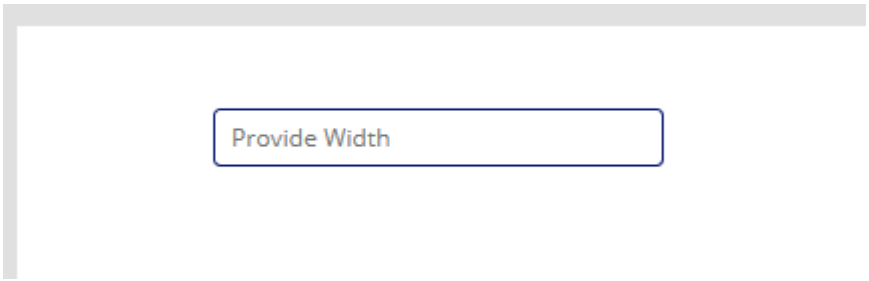
- Rename the Text Input **Width Text**.
- Remove the **Default** property of the **Width** text input.



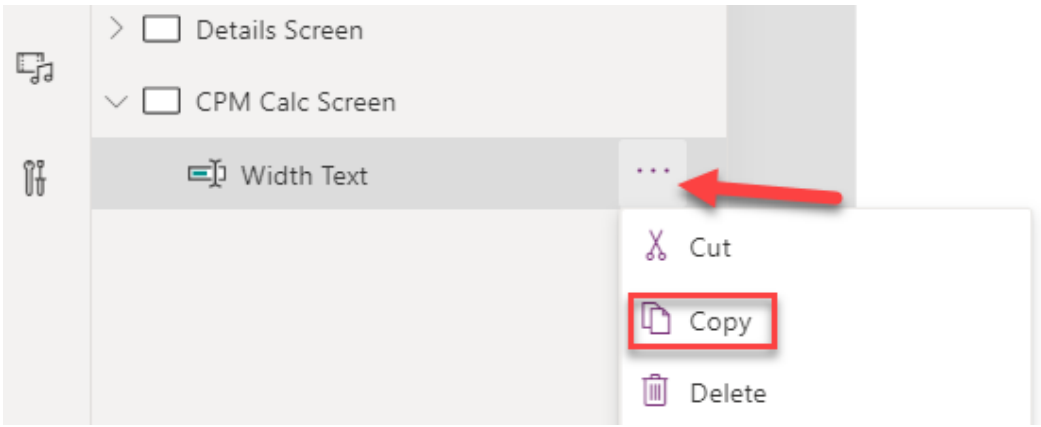
- Change the **HintText** property of the **Width** text input to **Provide Width**.



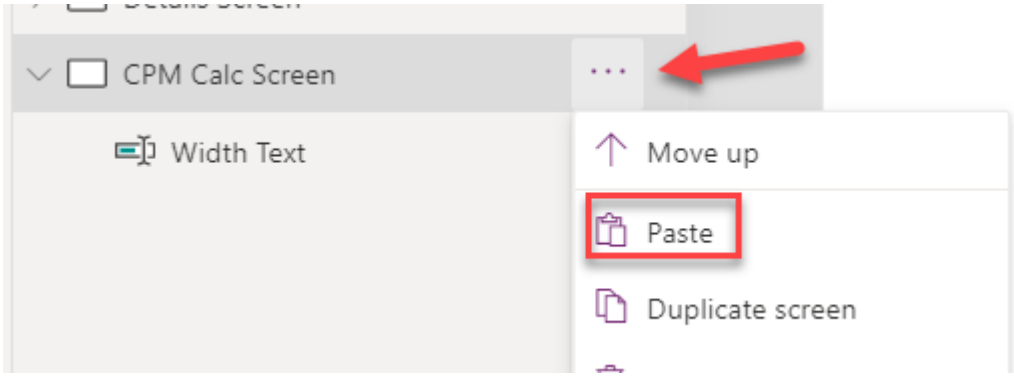
- The **Width Text** input should now look like the image below.



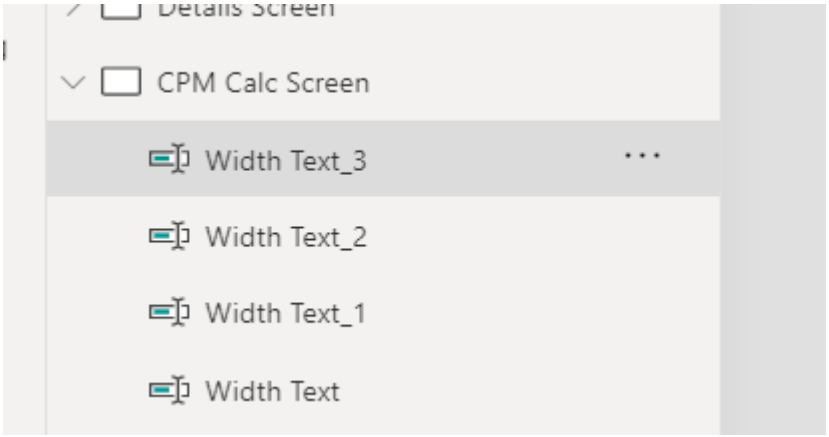
5. Add Height, Length, and Air Change Input Text controls
   - Copy the **Width Text**.



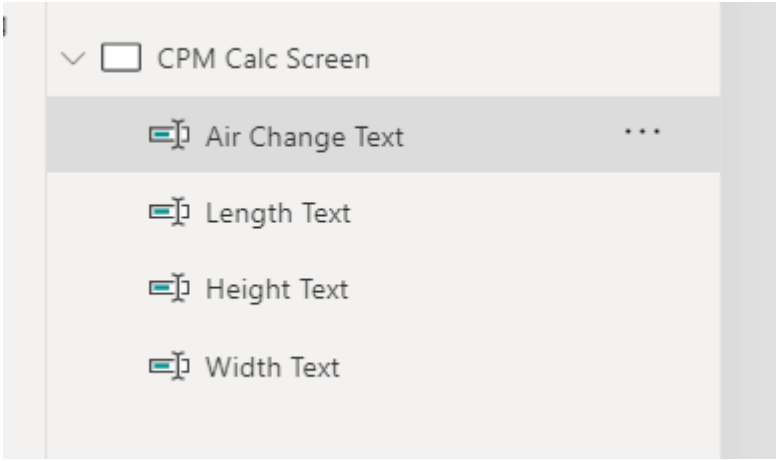
   - Paste the text input you copied to the **CPM Calc Screen.**



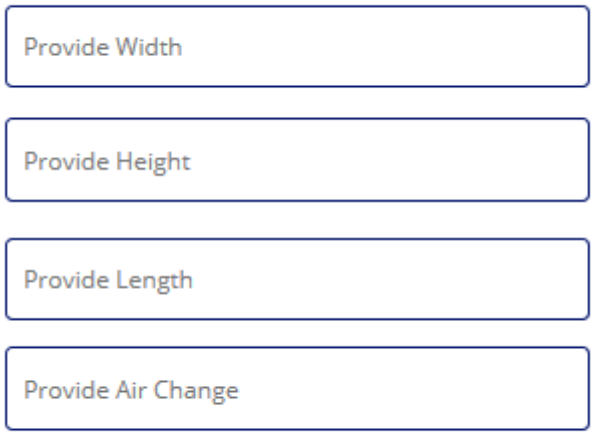
   - Paste the text input you copied to the **CPM Calc Screen** two more times.
   - The **CPMCalcScreen** should now have total of four text inputs.



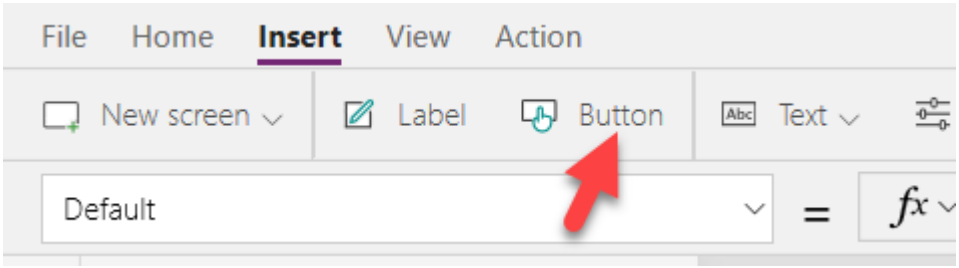
- Rename the input text controls **Height Text**, **Length Text**, and **Air Change Text**.



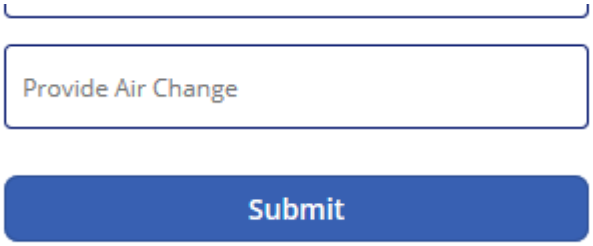
- Change the **HintText** for the three text inputs you renamed to **Provide Height**, **Provide Length**, and **Provide Air Change**, respectively.
- Resize and reposition the text inputs as shown in the image below.



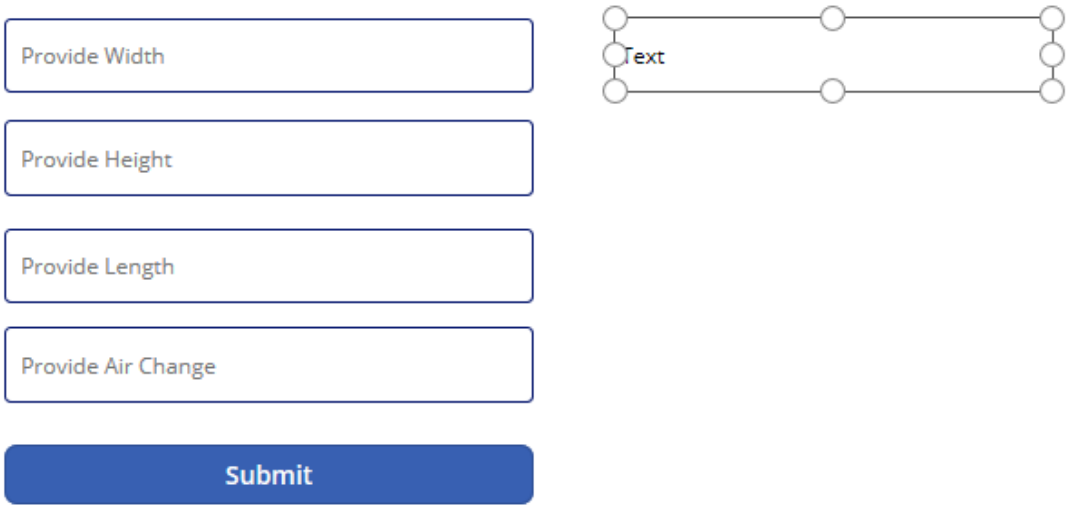
6. Add button
   - Go to the **Insert** tab and click Button.



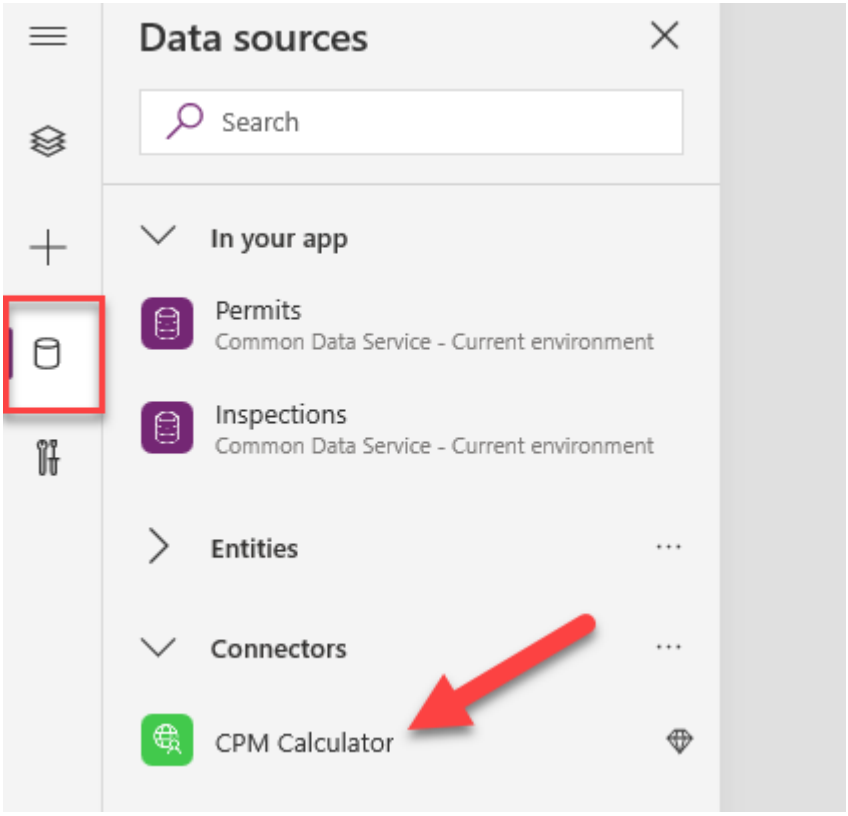
- Rename the Button **Calculate Button**.
- Change the **Calculate Button Text** value to **Submit**.
- Resize and reposition the button as shown in the image below.



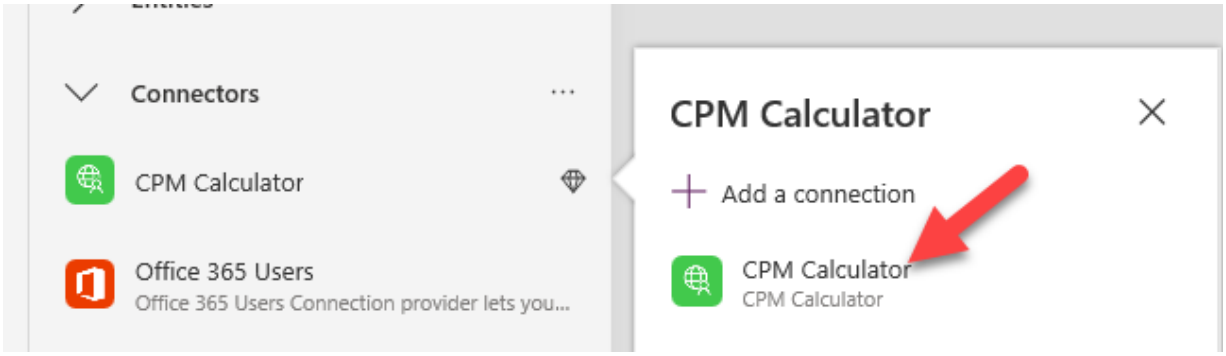
7. Add the result label to the screen
   - Go to the **Insert** tab and click Label
   - Rename the Label **Result Label**.
   - Place the label to the right of the text inputs.



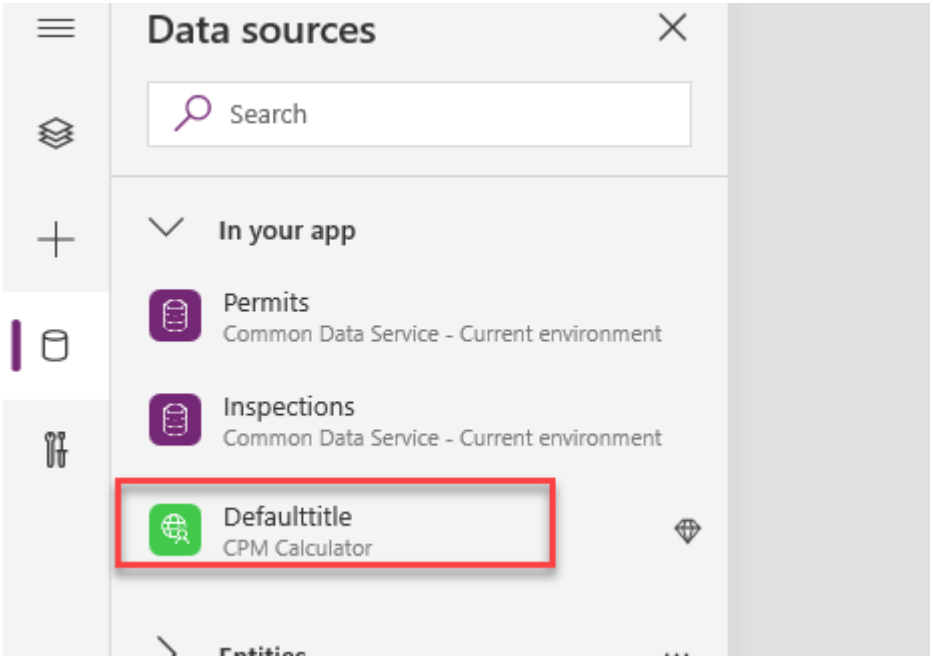
8. Add the Custom Connector to the application.
   - Select the **Data Sources** tab.
   - Expand **Connectors**.
   - Click the **CPM Connector**.



- Click **CPM Calculator** again.

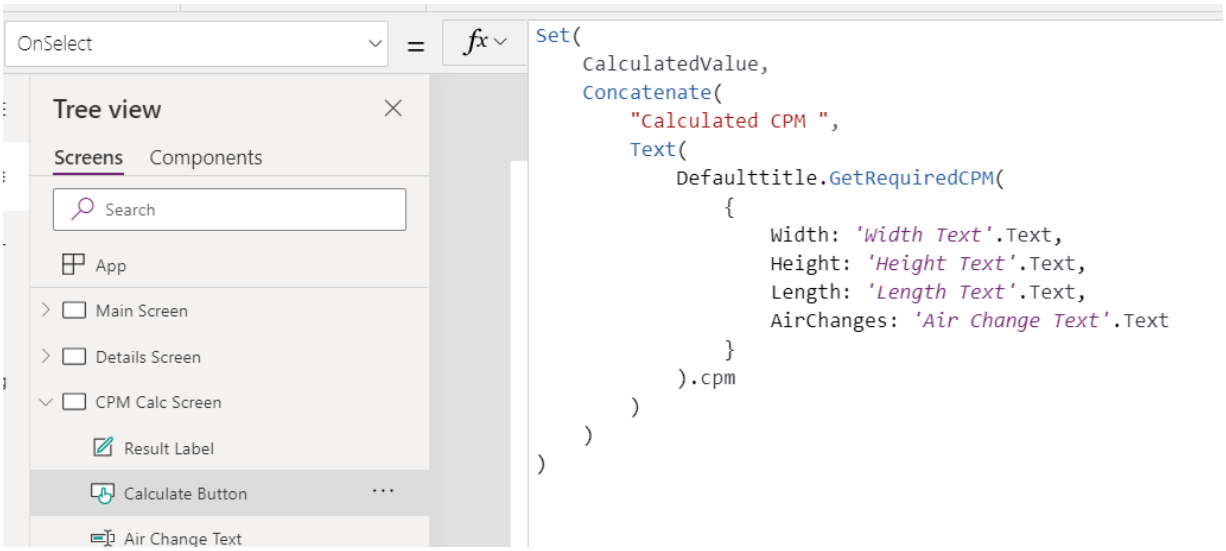


- The **CPM Calculator** should now be listed in the **In your App** section.

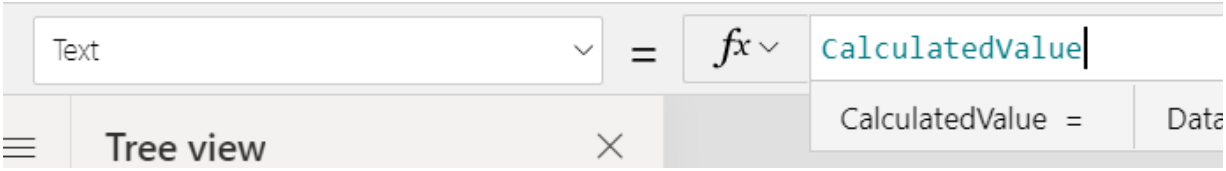


9. Get the calculated value when the button is clicked
   - Select the **Calculate Button**.
   - Set the **OnSelect** value of the **Calculate Button** to the formula below.

```
Set(CalculatedValue, Concatenate("Calculated CPM ", Text(Defaulttitle.GetRequiredCPM({Width:
```



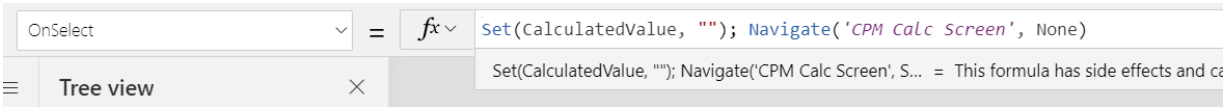
   - Select the **Result Label** and set the **Text** value to the **CalculatedValue** variable.



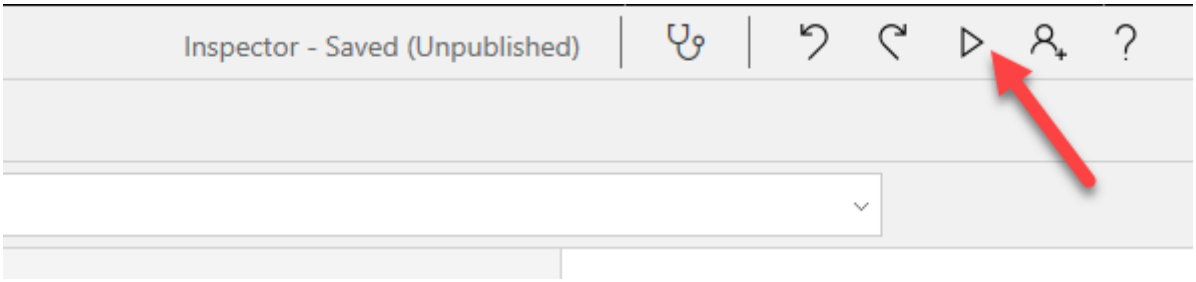
10. Add navigation button to the Main screen
    - Select the **Main Screen**.
    - Go to the **Insert** tab and click **Button**.
    - Rename the Button **CPM Button**.
    - Change the **CPM Button Text** value to **CPM Calculator**.
    - Place the button on the bottom right of the **Main Screen**.

11. Steps to navigate to the CPM Calc Screens
    - Select the **CPM Calculator**.
    - Set the **OnSelect** value of the **CPM Calculator** to the formula below.

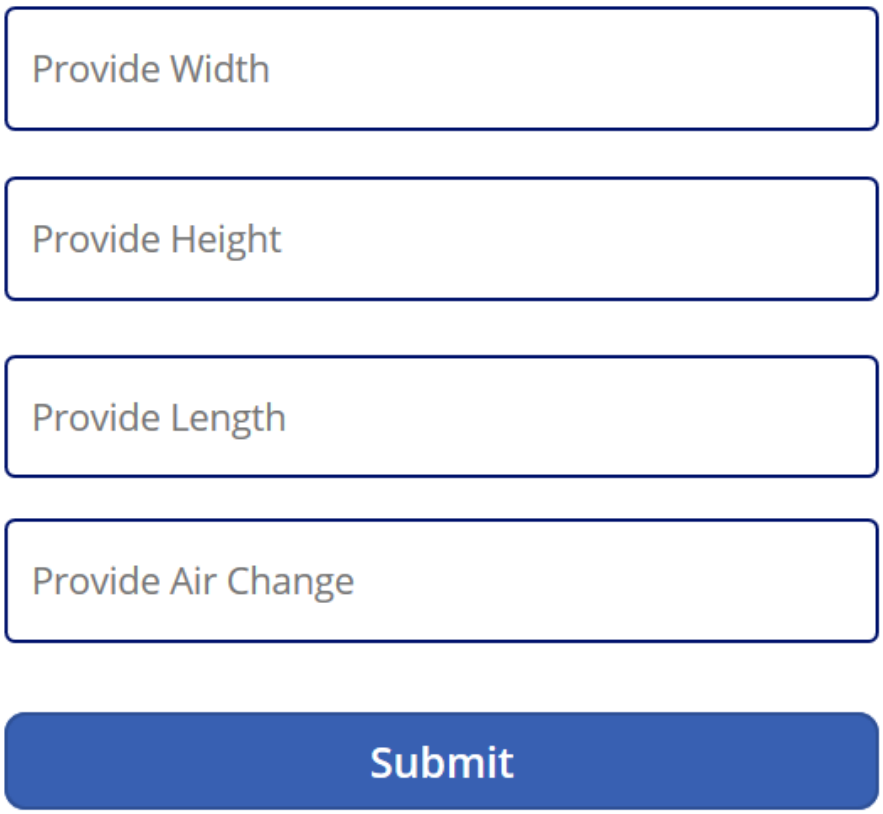
```
Set(CalculatedValue, ""); Navigate('CPM Calc Screen', ScreenTransition.None)
```



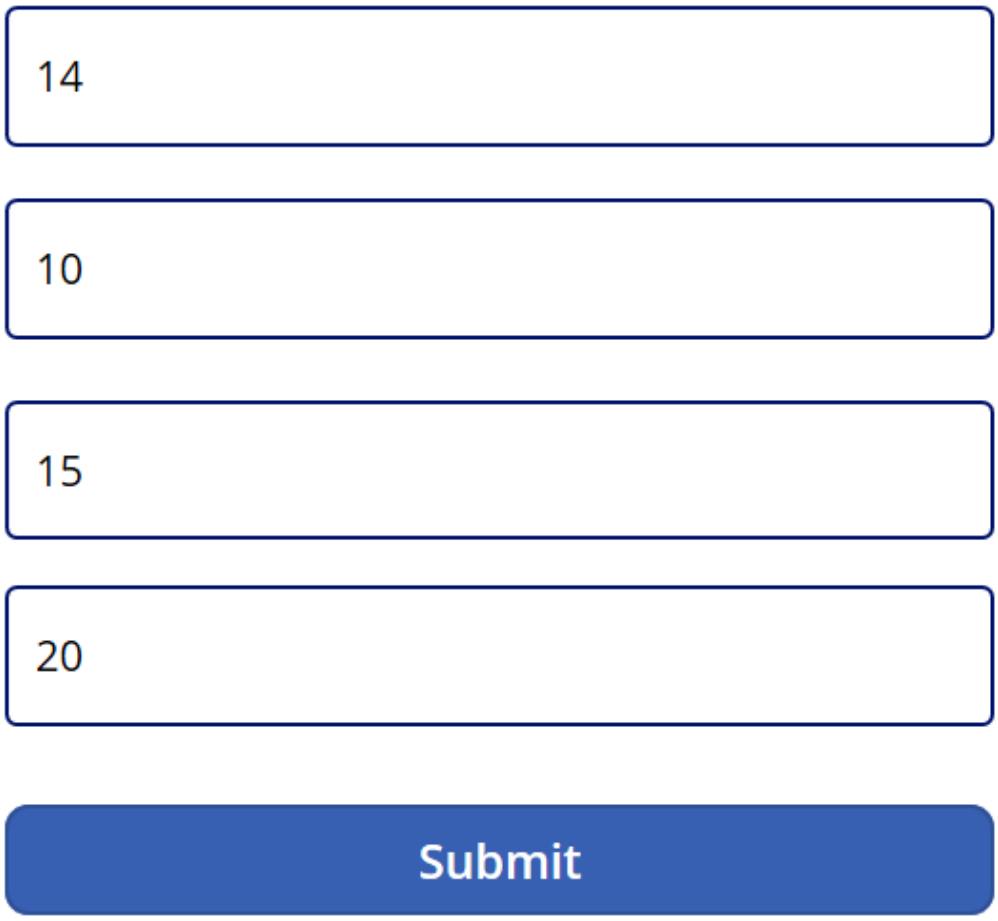
12. Run the Application
    - Select the **Main Screen** and click **Preview the App**.



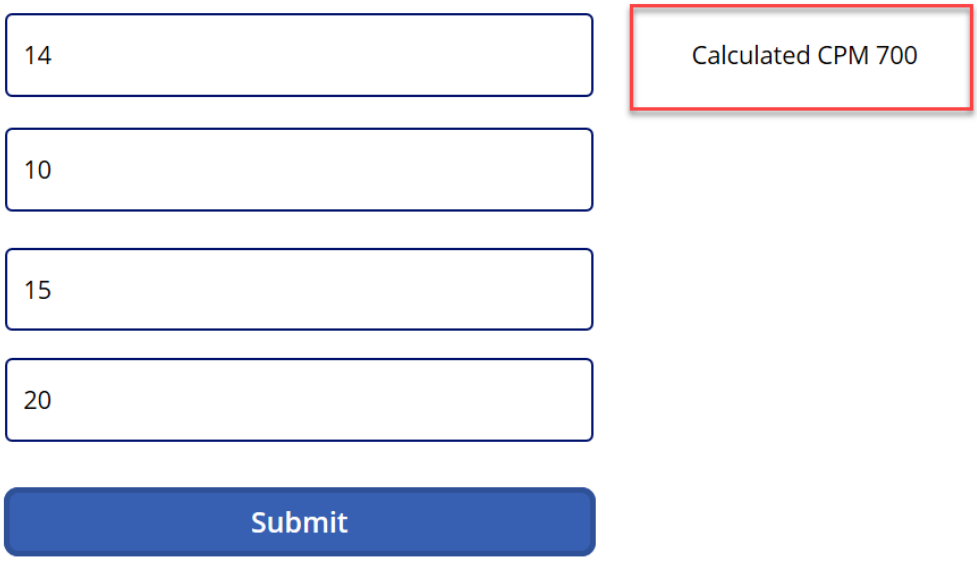
    - Click on **CPM Calculator** button.
    - The CPM Calculator screen should load.



    - Provide values and click **Submit**. You can notice the loading dots on top of the screen, which confirms that the request has been initiated.



- The **Result Label** should show the calculated result from the Custom Connector.



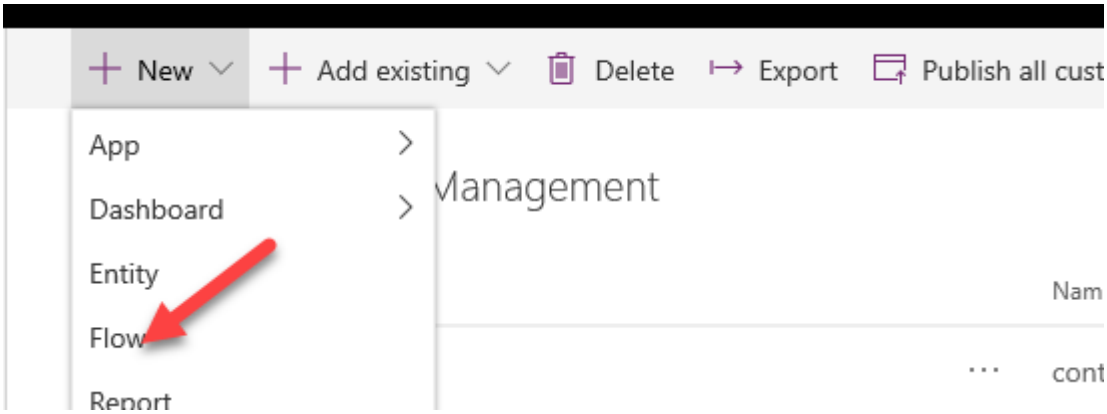
- Close the Preview.
- Click **File** and **Save**.

- **Publish** the changes to the application.
- Click **Close**.

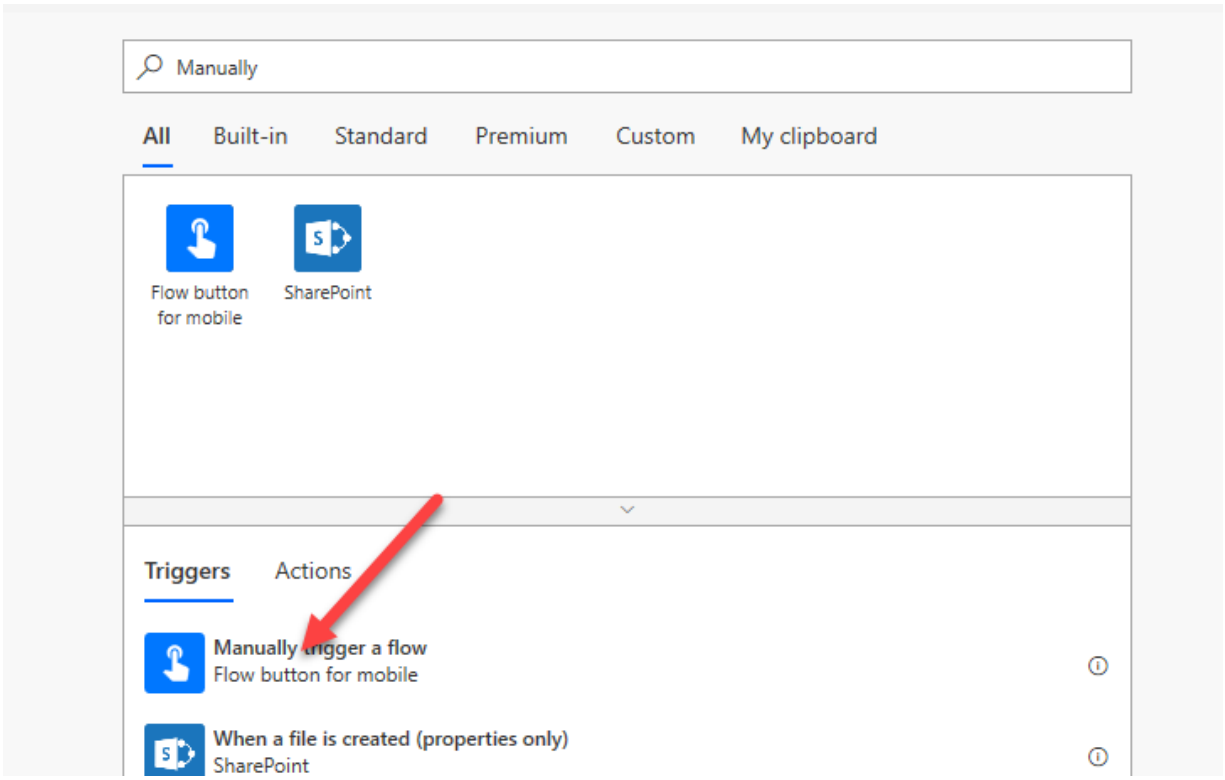
Click **Done** on the other window for the solution.

47.2 Task #2: Test on Flow

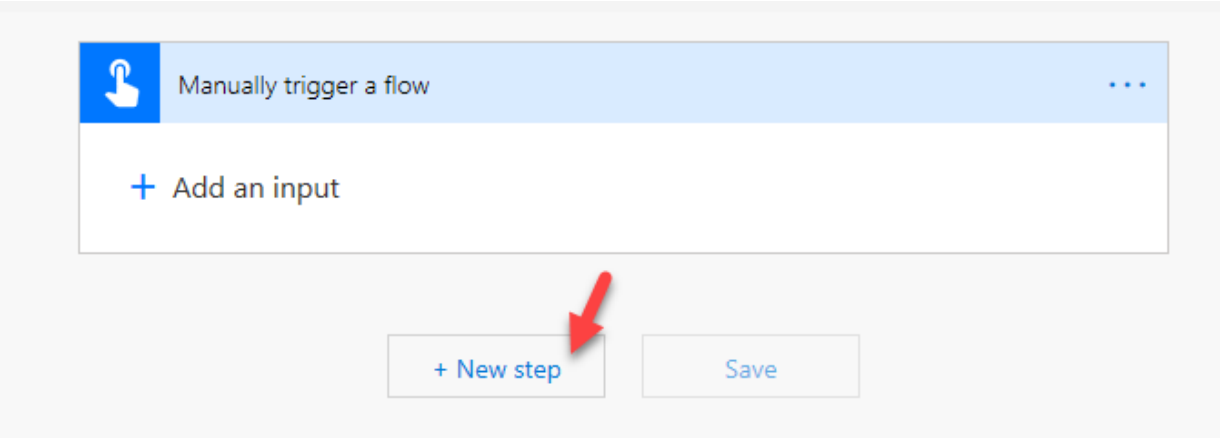
1. Open the Permit Management solution
   - Sign in to Power Apps maker portal and make sure you are in the **Dev** environment.
   - Select **Solutions**.
   - Click to open the **Permit Management** solution.
2. Create Flow and add trigger.
   - Click **New** and select **Flow**. This will open a New window.



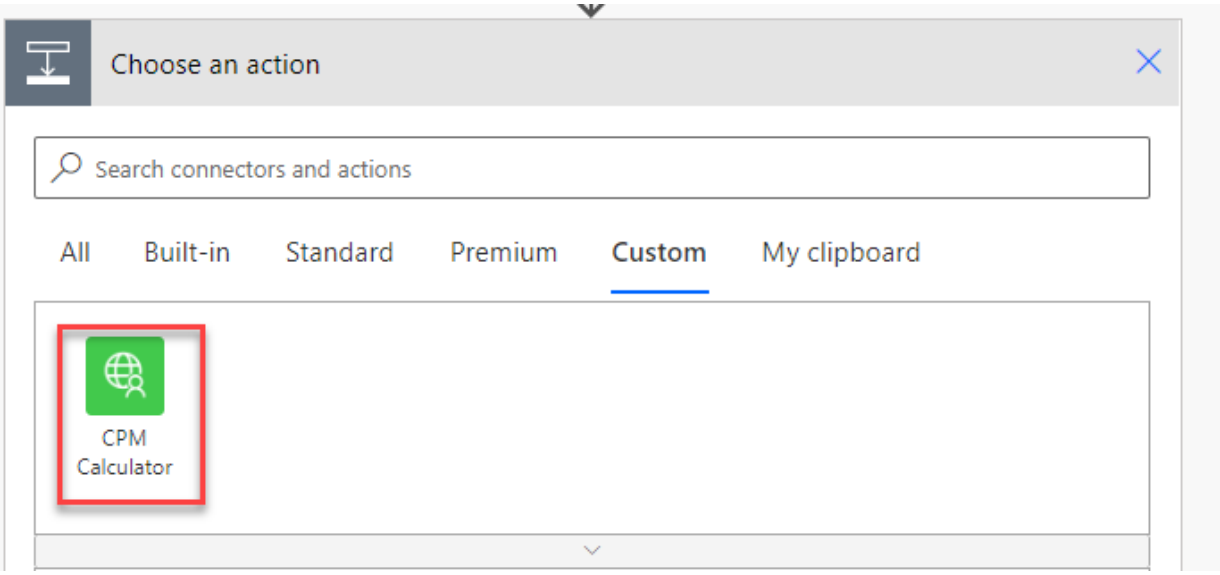
   - Search for **Manually** and select **Manually Trigger Flow**.



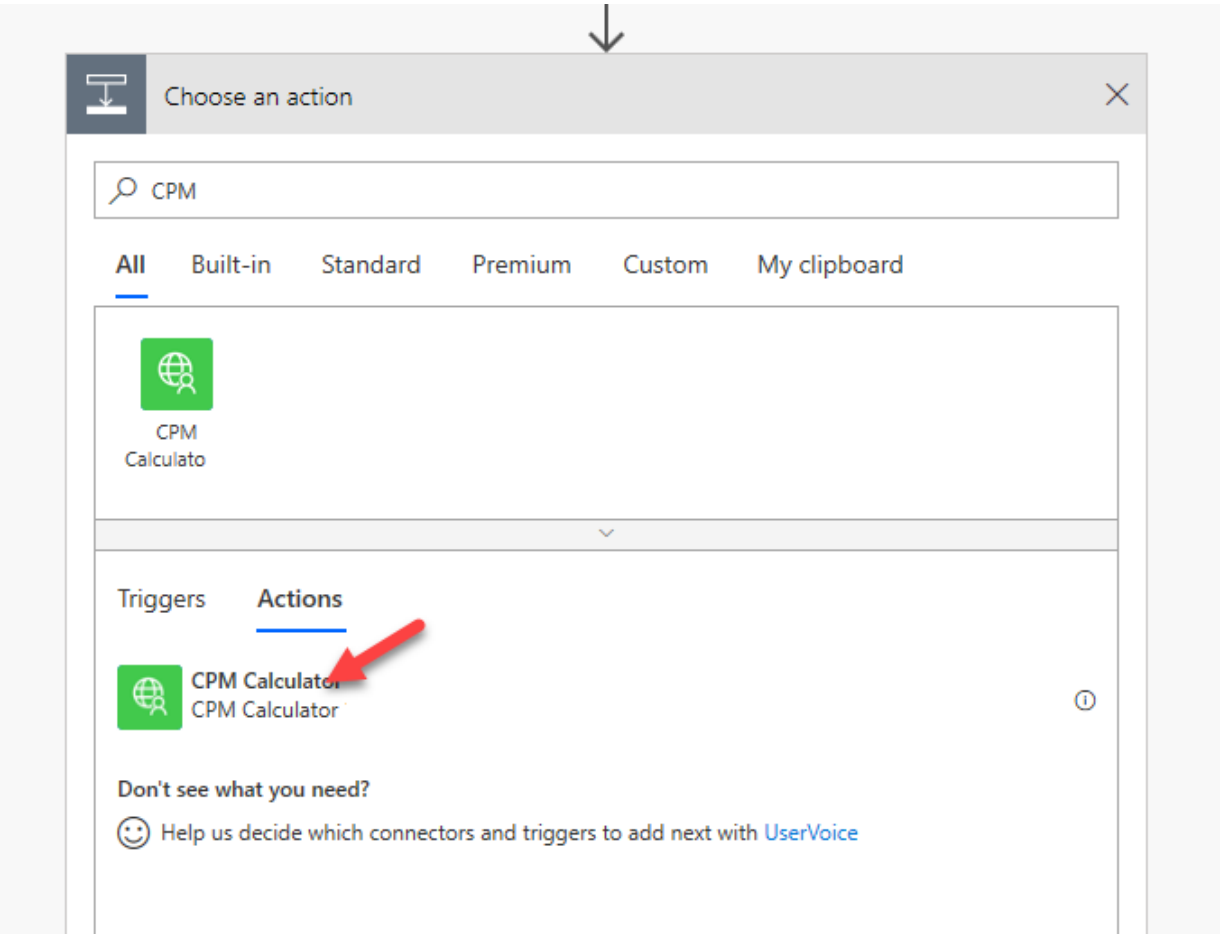
3. Add a step that will use the Custom Connector
   - Click + **New Step**.



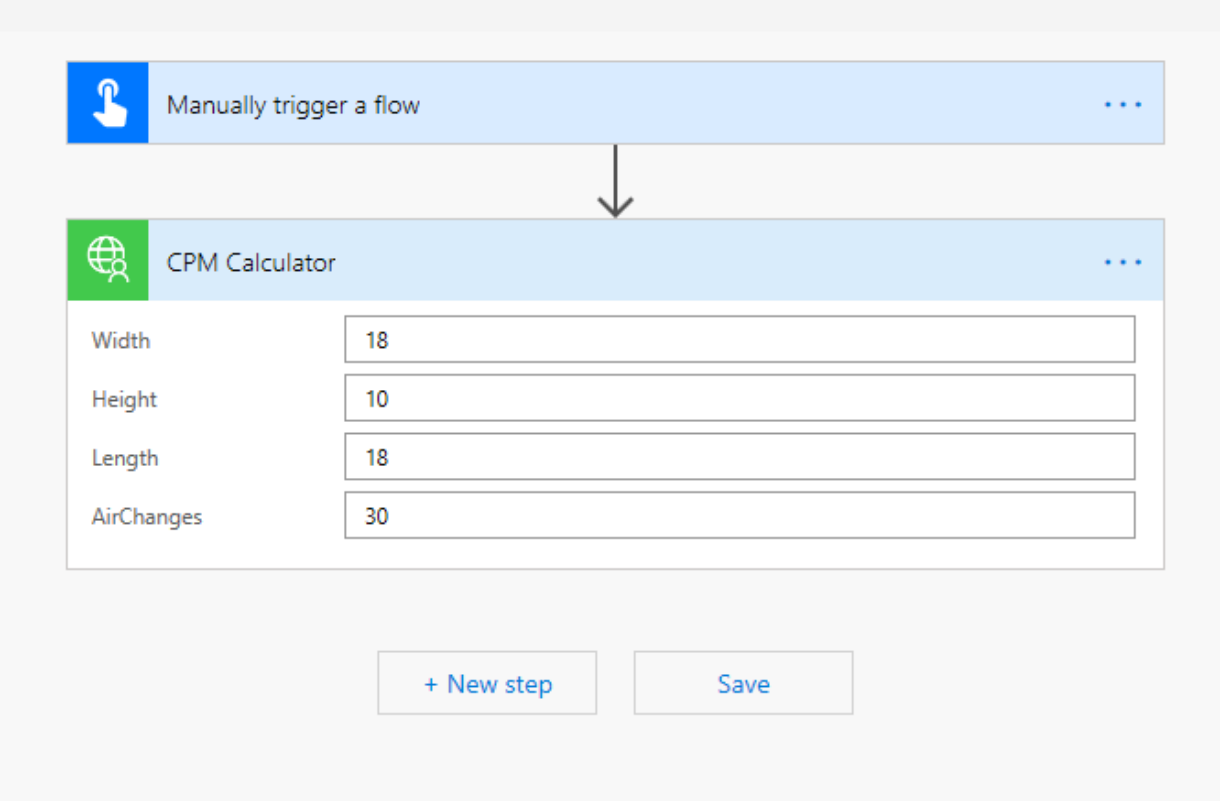
- Select the **Custom** tab and click **CPM Calculator**.



- Select **CPM Calculator** action.

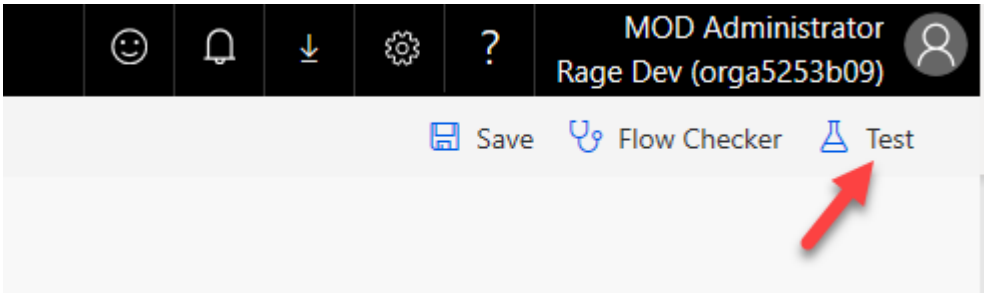


4. Provide values and save
   - Enter 18 for Width, 10 for Height, 18 for Length, 30 for AirChanges, and click **Save**.

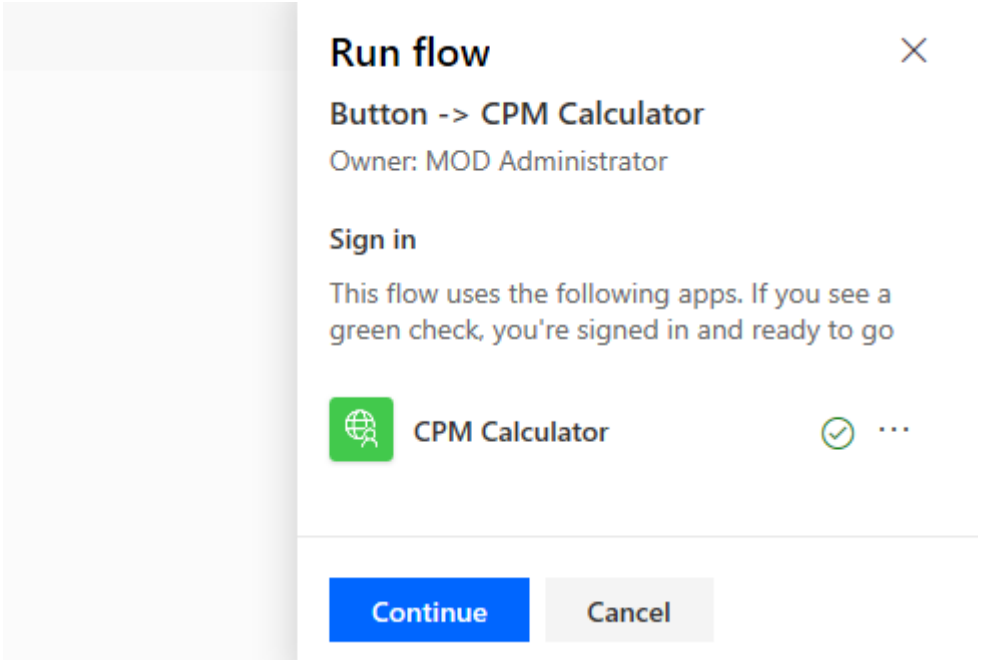


5. Test the Flow

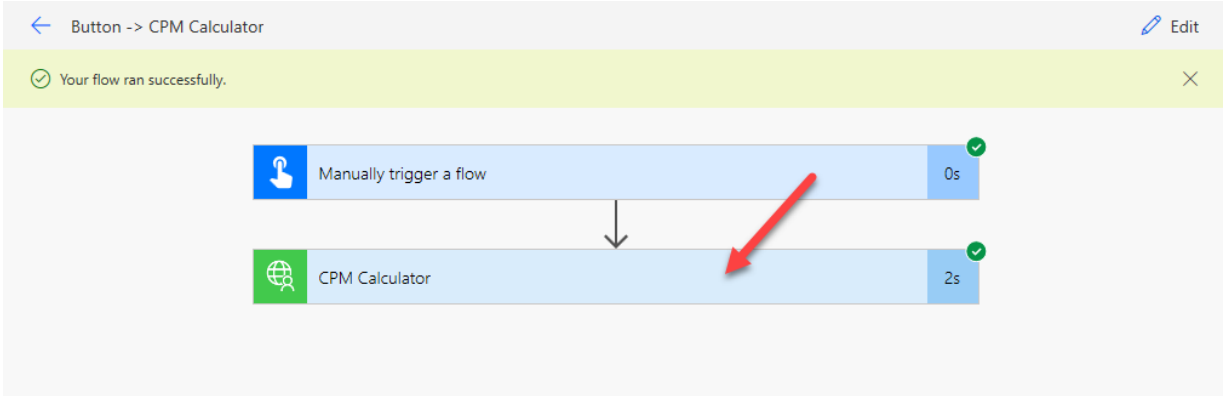
- Click **Test**.



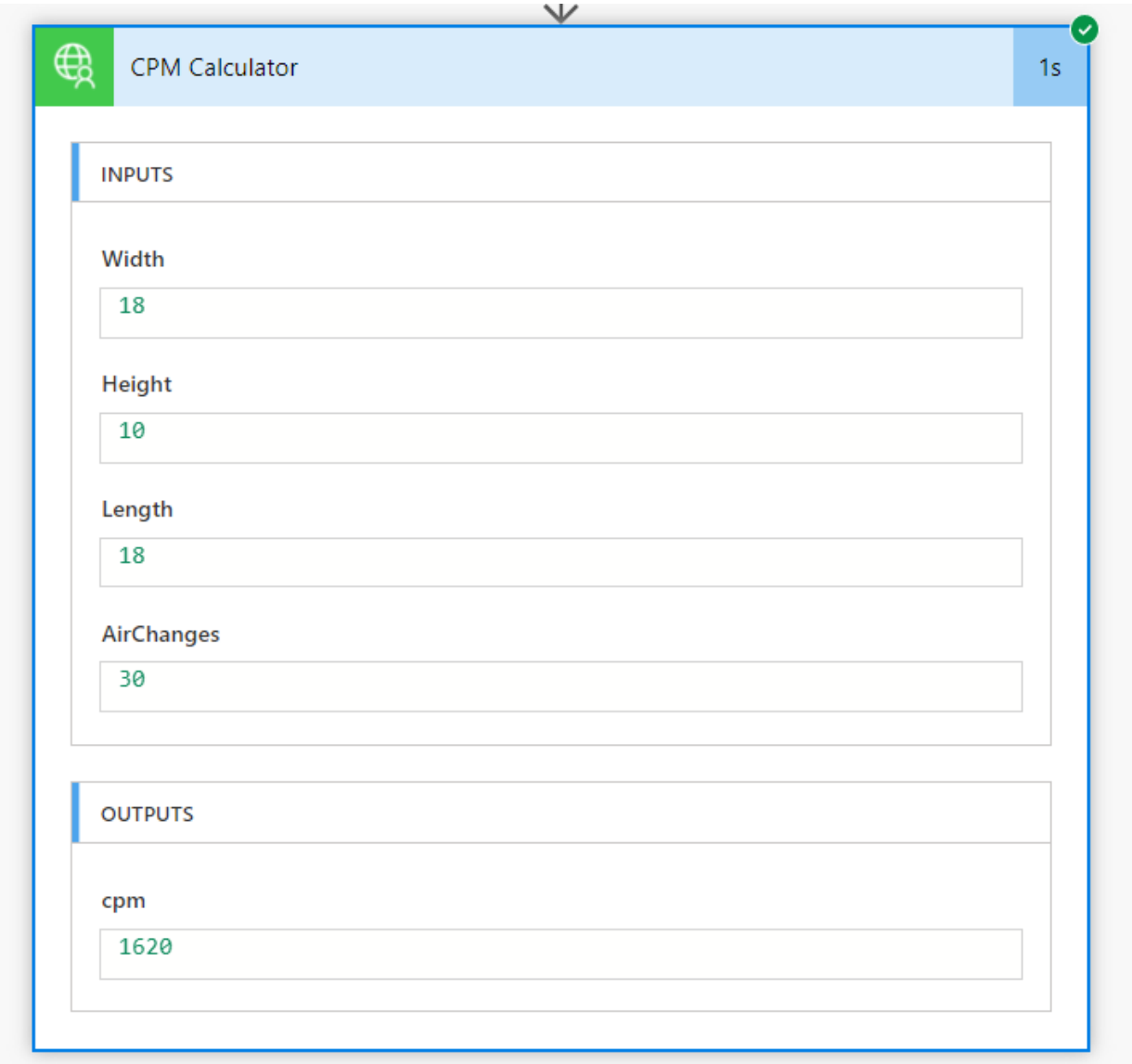
- Select **I will Perform the Trigger** and click **Save & Test**.
- Click **Continue**.



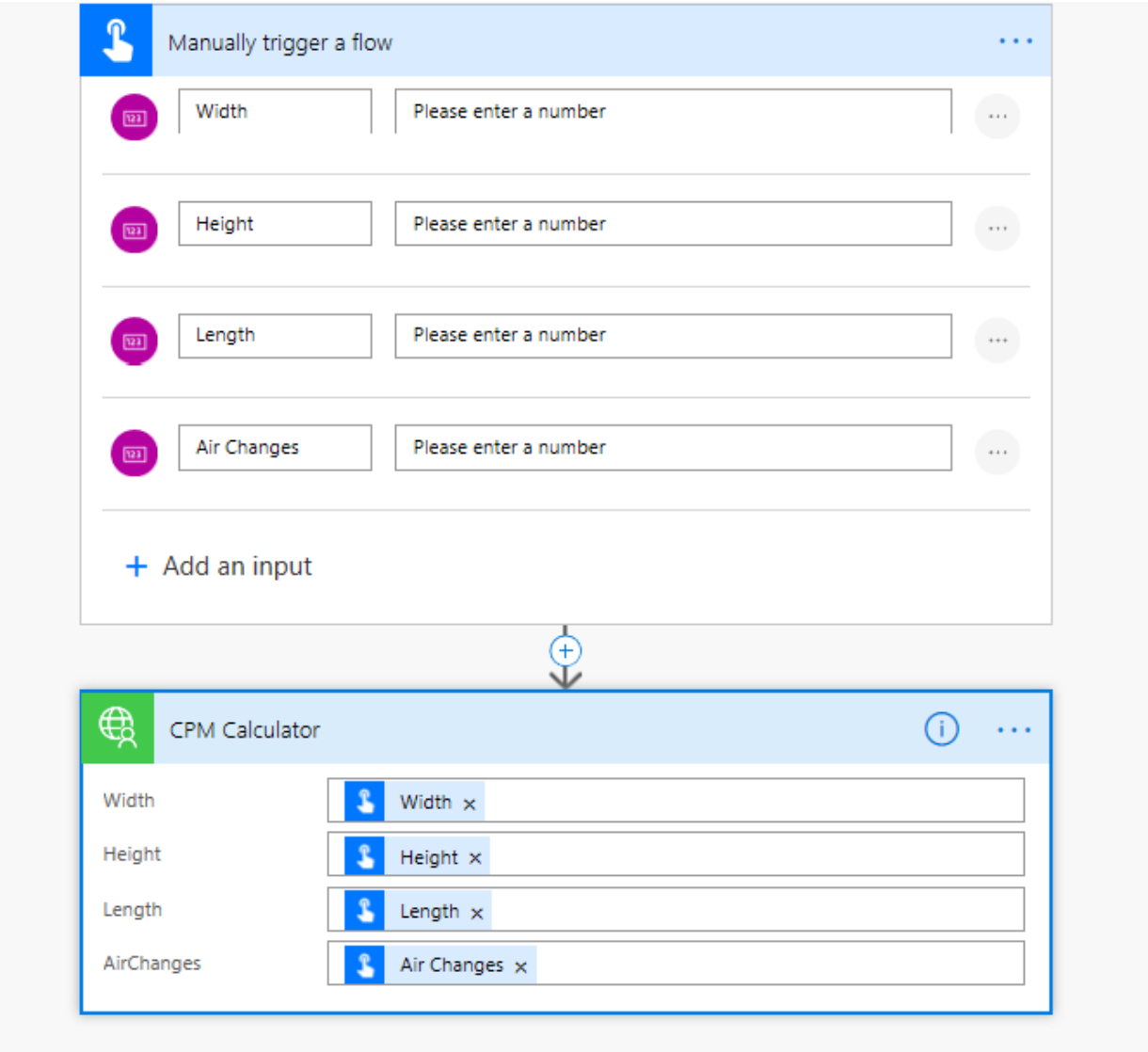
- Click **Run Flow**.
- Click **Done.** The Flow should run successfully. In the Flow run history, expand the CPM Calculator action.



- You should be able to see the calculated result from the custom connector in the output of the action.



- Close the window and go back to Solution window. Click **Done.**
- Click **Publish all Customizations.**
- If you finish early, try adding input values to the Manual Button trigger for the room dimensions and use those to call the custom connector. You could also use the notification connector to send the user the required CPM. Finally, if you want to test this in a real device install the Power Automate mobile application.



47.3 lab: title: 'Lab 10: Application Lifecycle Management'

[!NOTE] Effective November 2020:

- Common Data Service has been renamed to Microsoft Dataverse. Learn more
- Some terminology in Microsoft Dataverse has been updated. For example, *Table* is now *table* and *field* is now *column*. Learn more

This content will be updated soon to reflect the latest terminology.

47.4 Lab 10 – Application Lifecycle Management

48 Scenario

A regional building department issues and tracks permits for new buildings and updates the remodeling of existing buildings. Throughout this course you will build applications and automation to enable the regional building department to manage the permitting process. This will be an end-to-end solution which will help you understand the overall process flow.

In this lab you will use Azure DevOps for source control of your solution assets. As you have been building your app you have been tracking all the changes in a Permit Management solution. You have exported this solution, so you had a back up copy. You have also manually imported the managed version of the solution into your production environment. As part of this lab you will see how you can automate working with solutions and use the Power Apps Azure DevOps tasks to check the changes into an Azure DevOps Repository. This

is the start of an overall ALM process that you would put in place to automate the complete lifecycle form development to production using Azure DevOps automation. In this lab you will be completing the first phase of that automation.

49 High-level lab steps

As part of configuring Azure DevOps ALM automation, you will complete the following

- Sign in to your Azure DevOps account
- Create an Azure DevOps project
- Configure the Power Apps ALM tasks
- Build an export solution pipeline
- Test the export from dev to Azure DevOps

49.1 Things to consider before you begin

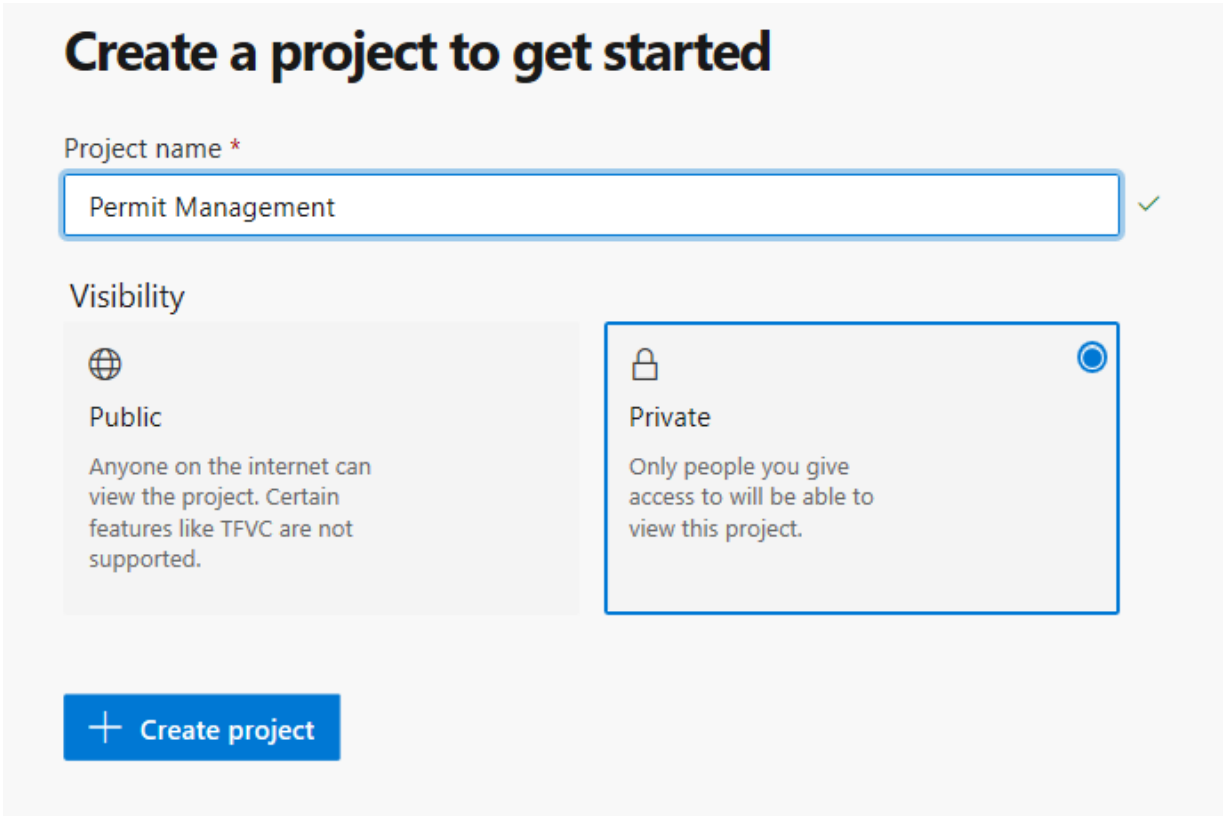
- How often are you planning to run the build process?
- Is it going to be fully automated or run manually?
- How many users will be committing changes into the repositories and how often?
- How many instances are you planning to control?
- Are there any other build tasks you should consider?

50 Exercise #1: Initialize Azure DevOps

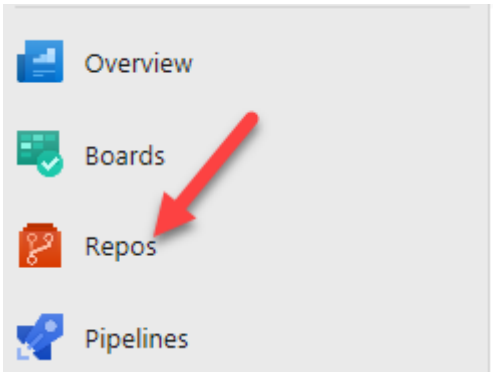
Objective: In this exercise, you will use your Azure DevOps account to track the solution assets of the Permit Management app.

50.1 Task #1: Sign in for Azure DevOps

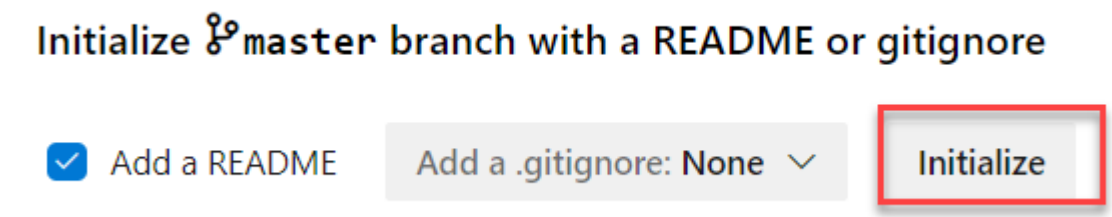
1. Sign in for Azure DevOps
    - Navigate to Azure DevOps
    - Provide your admin credentials and sign in.
2. Create the Azure DevOps project
    - Enter **Permit Management** for **Project Name**, select **Private**, and click **Create Project**.



3. Initialize Repository
   - Select **Repos**.

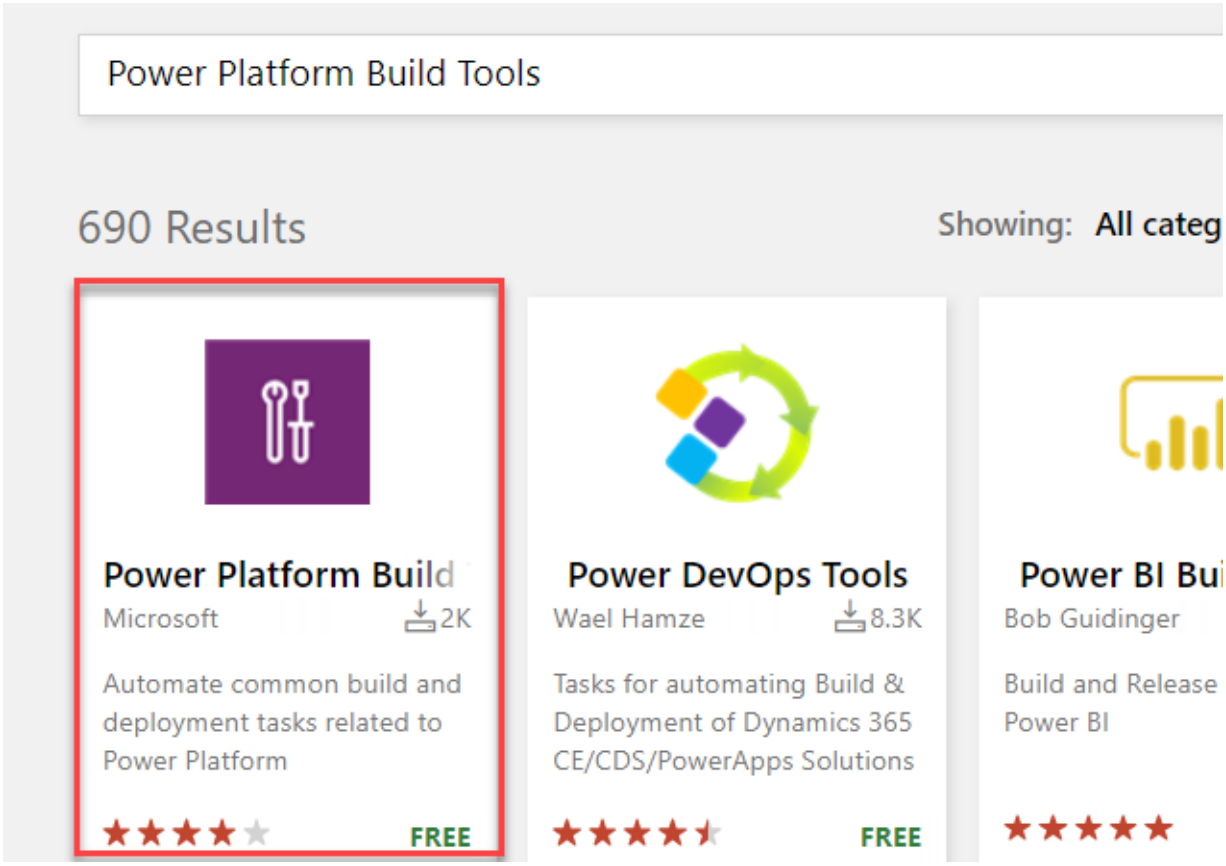


   - Scroll down to the bottom, check the **Add a Readme** checkbox, and click **Initialize**.

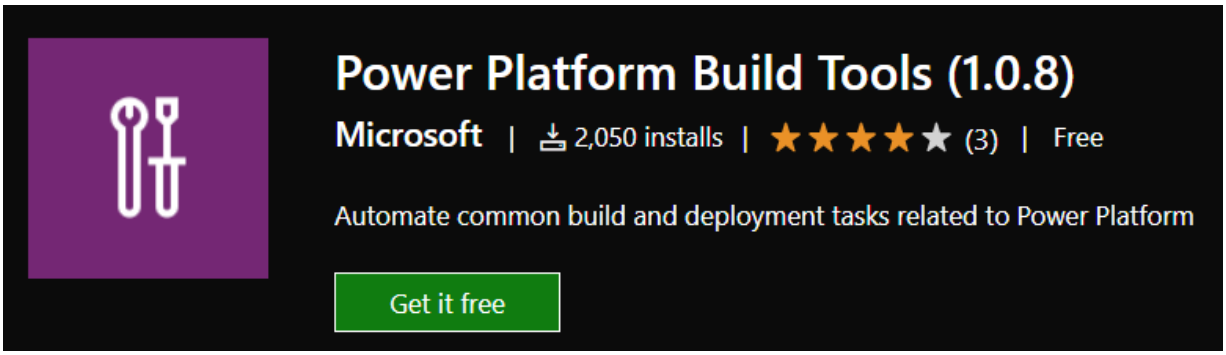


50.2 Task #2: Configure Power Apps ALM Tasks

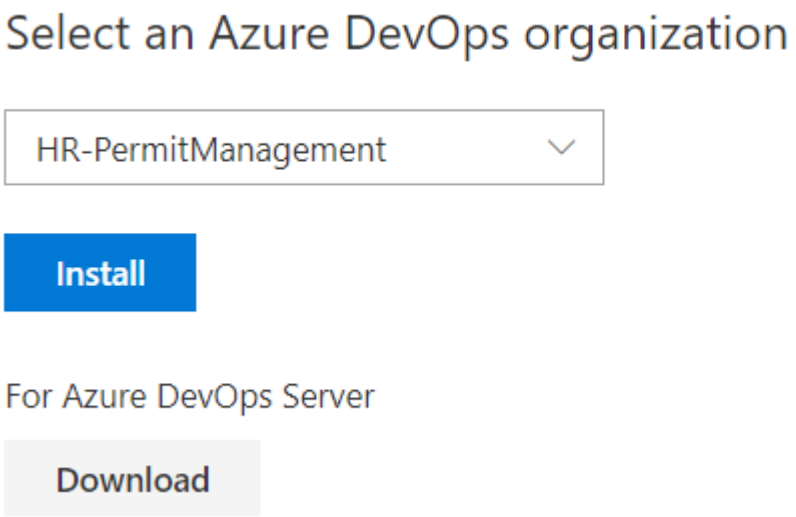
1. Get Power Apps BuildTools
   - Sign in to Visual Studio marketplace
   - Search for **Power Platform Build Tools**.
   - Select **Power Platform Build Tools**.



- Click **Get it Free**.

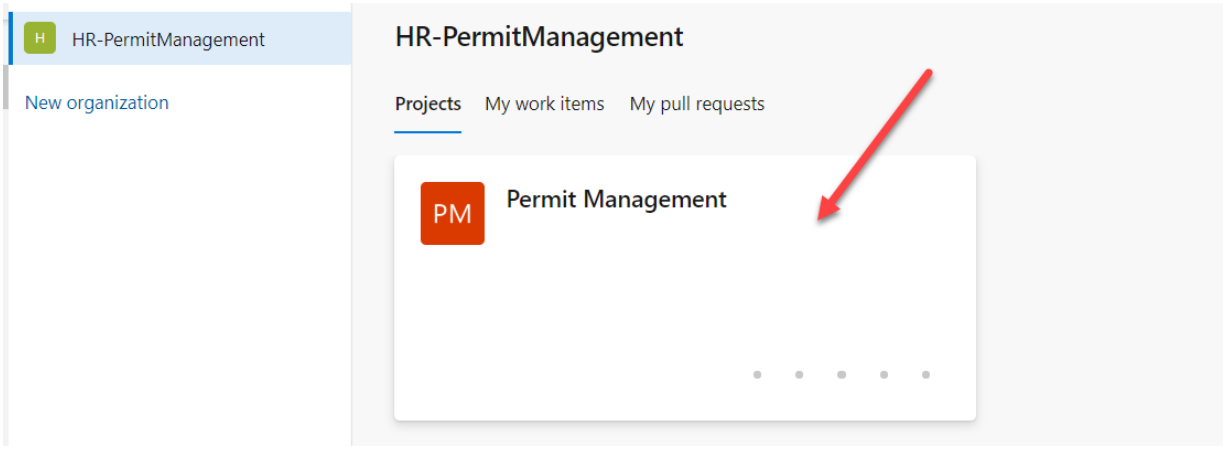


- Select the **Azure DevOps** organization you created and click **Install**.

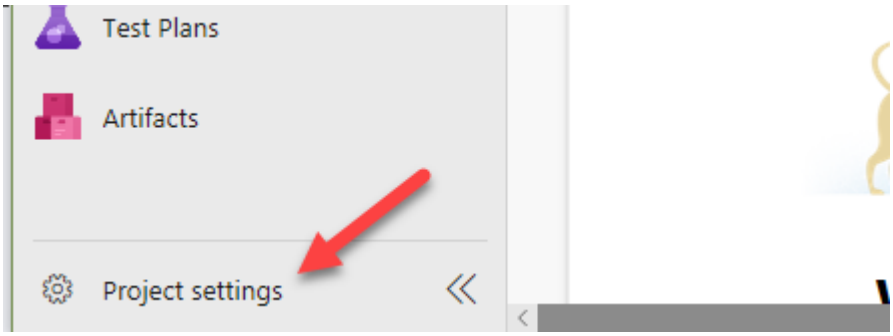


- Click **Proceed to Organization**.

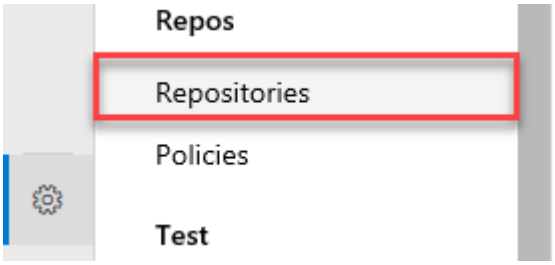
2. Go to Git repositories security
   - Click to open the **Permit Management** project you created.



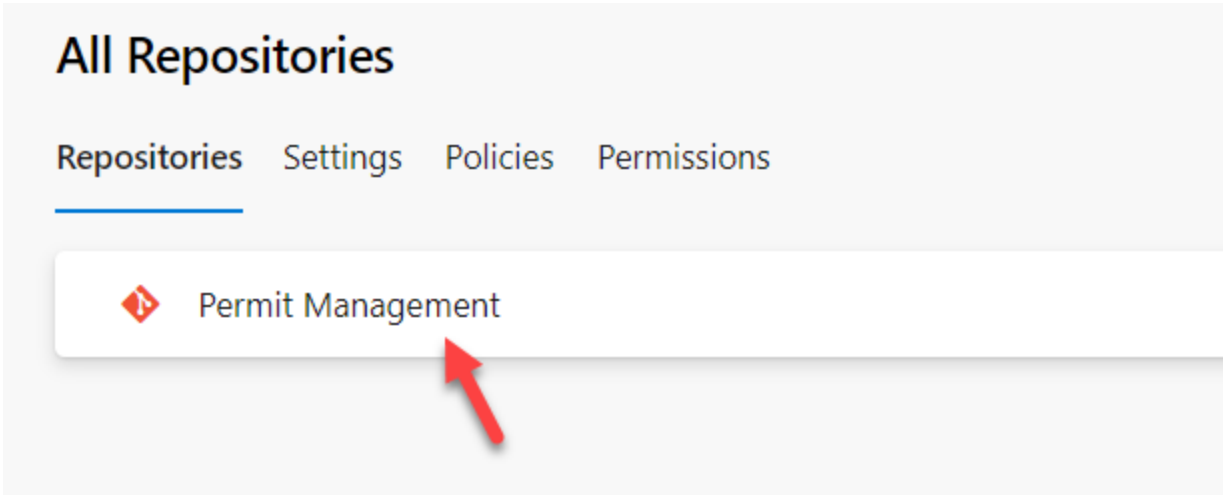
   - Click **Project Settings**.



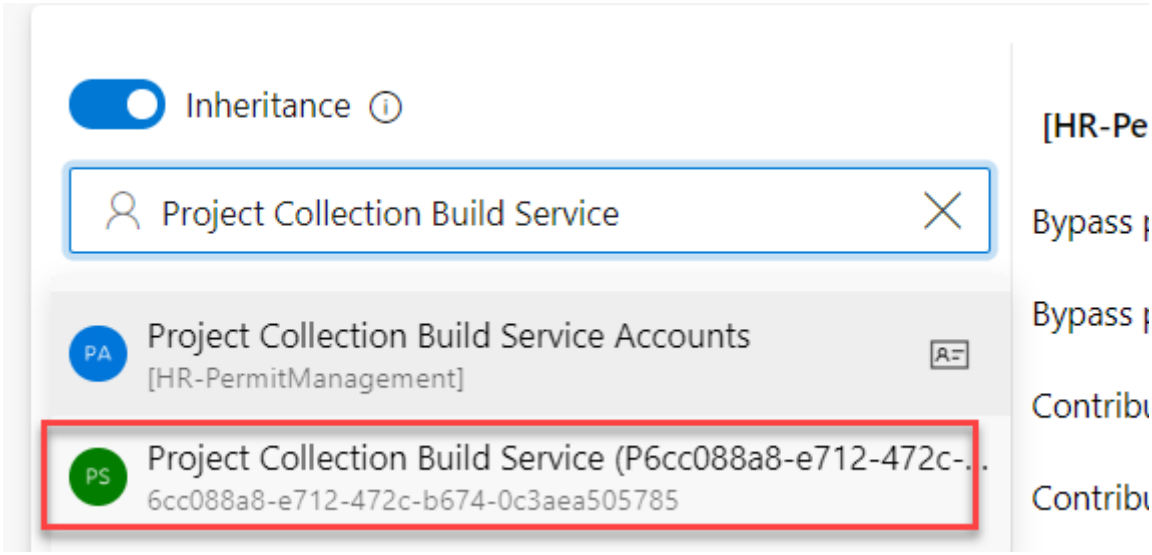
   - Select **Repositories**.



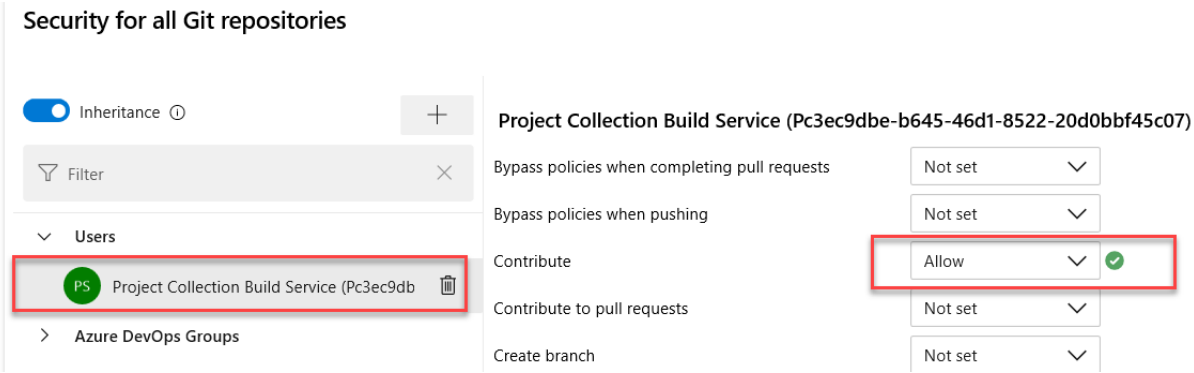
3. Add permissions to allow the build account to check in solution assets
   - Select the **Permit Management** project.



- Select the **Permissions** tab.
- Search for **Project Collection Build Service** and select the one without the accounts **Project Collection.**



4. Set Contribute permission for the Service Accounts
   - Select the **Project Collection Build Service**.
   - Locate the **Contribute** permission and select **Allow**.



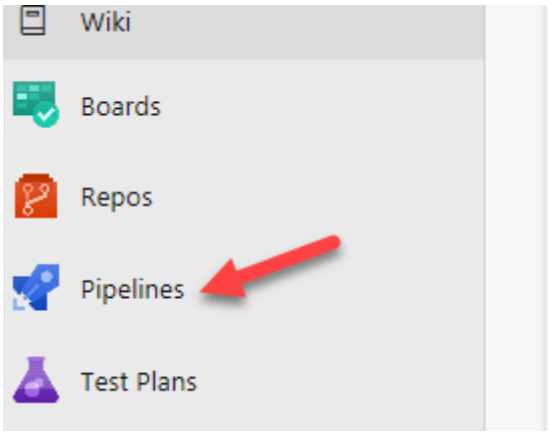
   - Click **Show More** to expand the menu.

51 Exercise #2: Build Export Pipeline

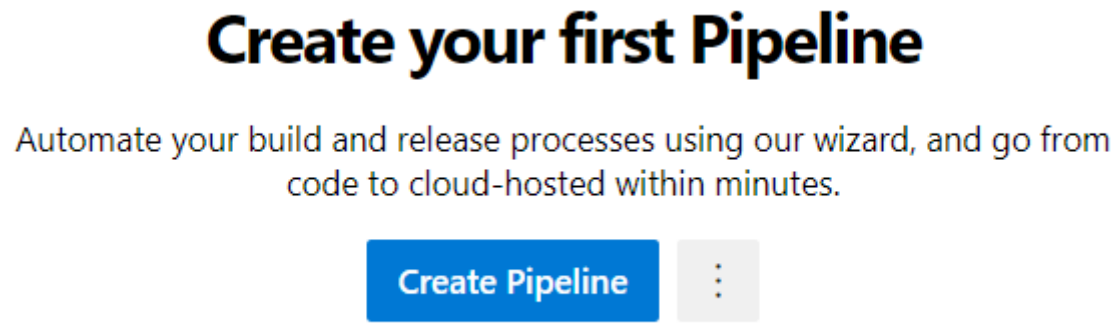
Objective: In this exercise, you will build an Azure DevOps pipeline that will export the solution from the development Microsoft Dataverse environment, unpack the solution file to individual files and then check those files into the repository.

51.1 Task #1: Export the Solution

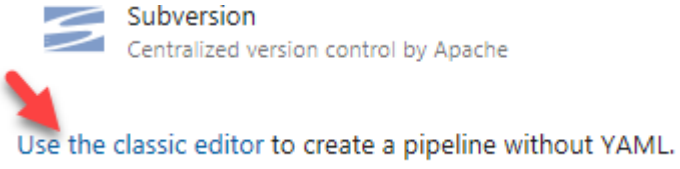
1. Create Build Pipeline
   - Click to expand **Pipelines**.



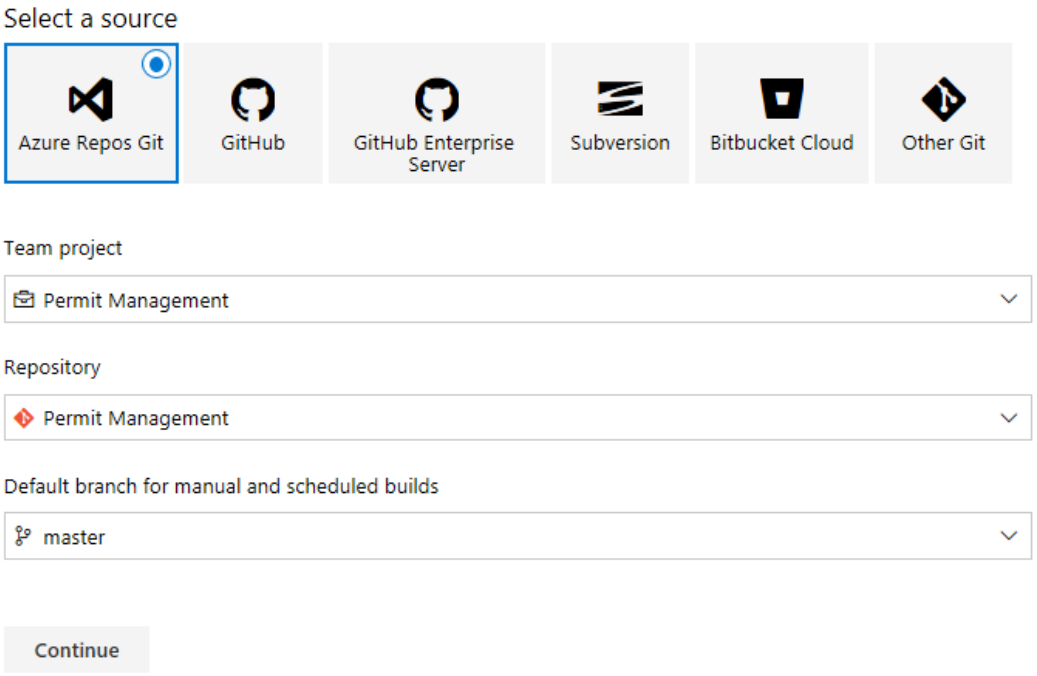
   - Click **New Pipeline**.



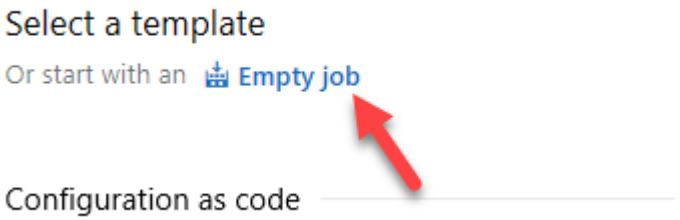
   - Click **Use the Classic Editor.**



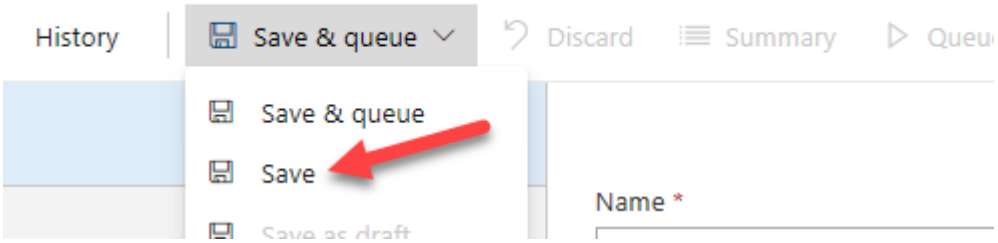
   - Don't change the default values and click **Continue**.



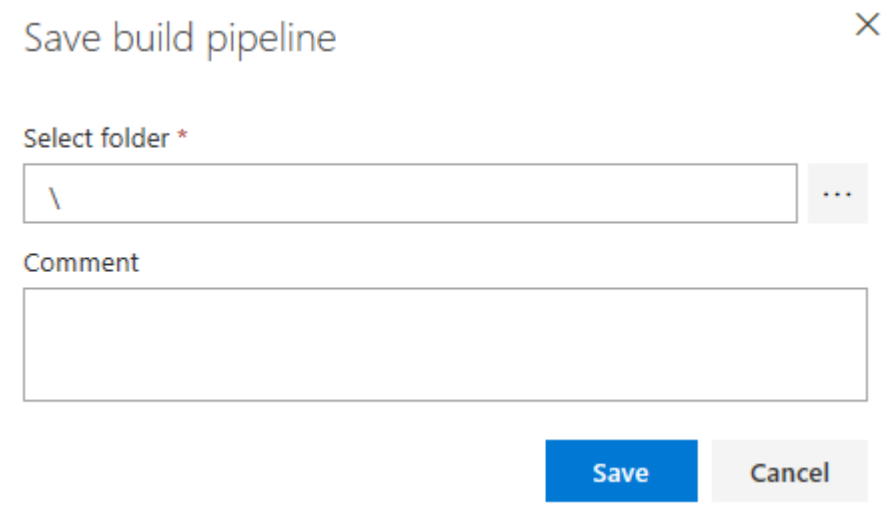
- Select **Empty Job**.



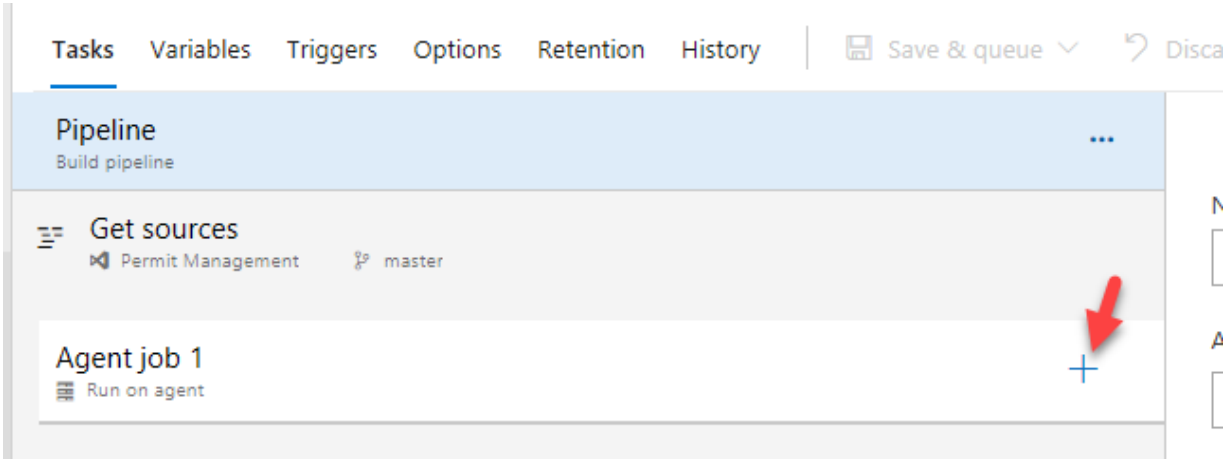
- Click **Save and Queue** and select **Save**.



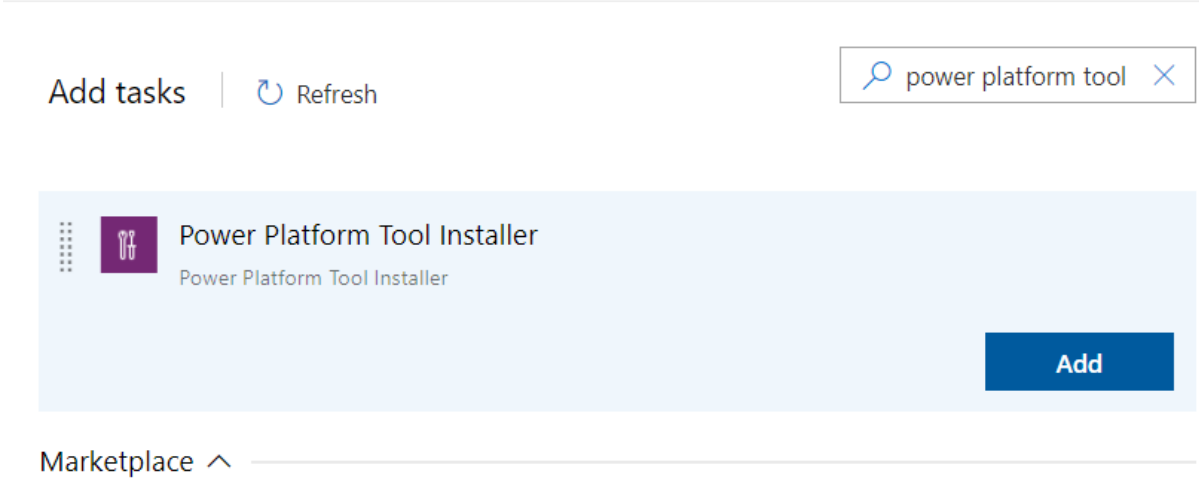
- Click **Save**.



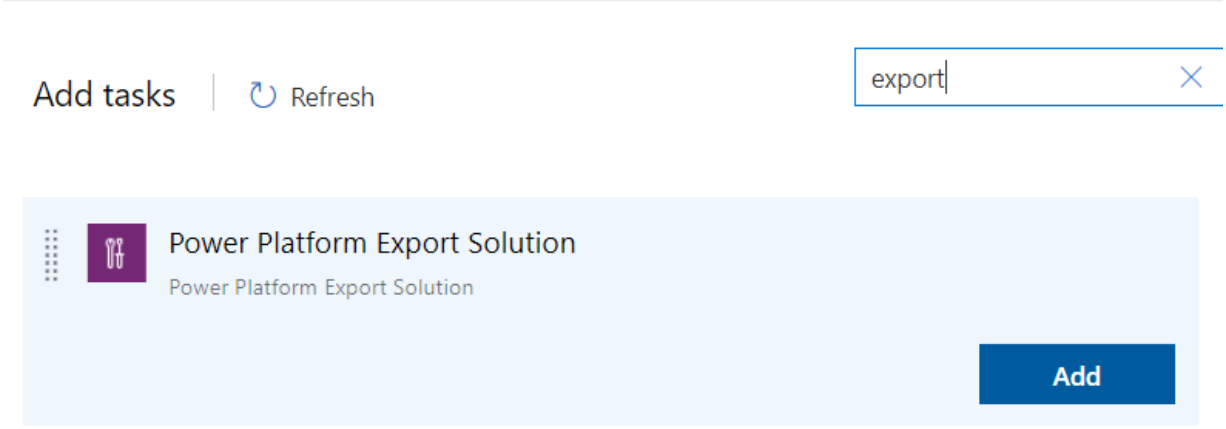
2. Add Power Apps Tool Installer task
   Note: The Power Apps Tool Installer needs to be run before any other Power Apps ALM tasks.
   - Click + icon to add Task to **Agent Job 1**.



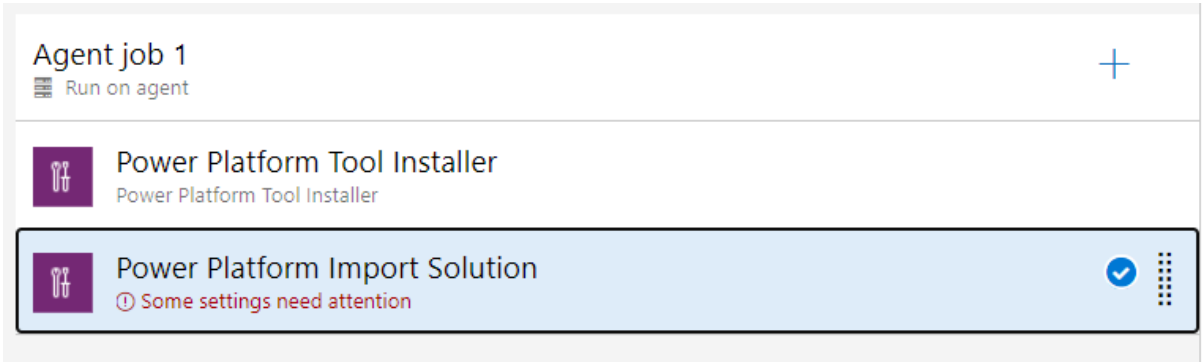
   - Search for **Power Platform Tool,** hover over select **Power Platform Tool Installer** and click **Add**.



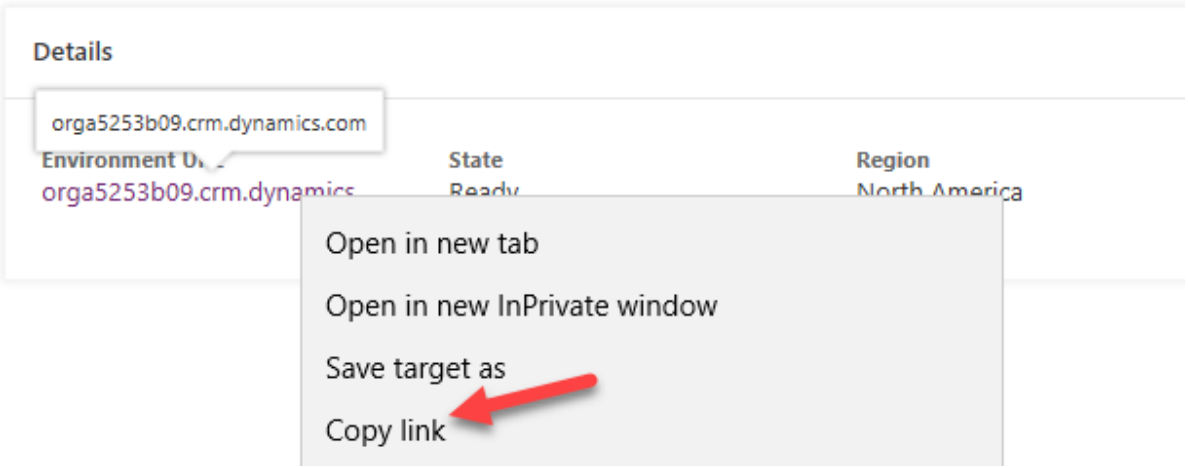
3. Add PowerApps Export Solution task
   - Search for **Export.**
   - Hover over **Power Platform Export Solution** and click **Add**.



4. Open PowerApps Export Solution
   - Select the **Power Platform Export Solution** task.

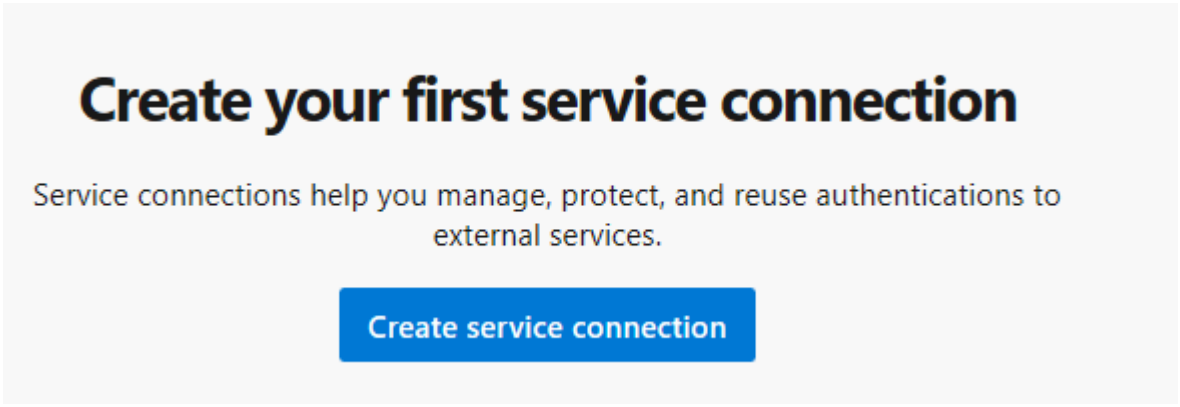


5. Get your Dev Environment URL
   - Start a new browser window or tab and sign in to Power Platform admin center
   - Select **Environments** and click to open the **Dev** environment.
   - Copy the **Environment URL** and keep it in your clipboard.

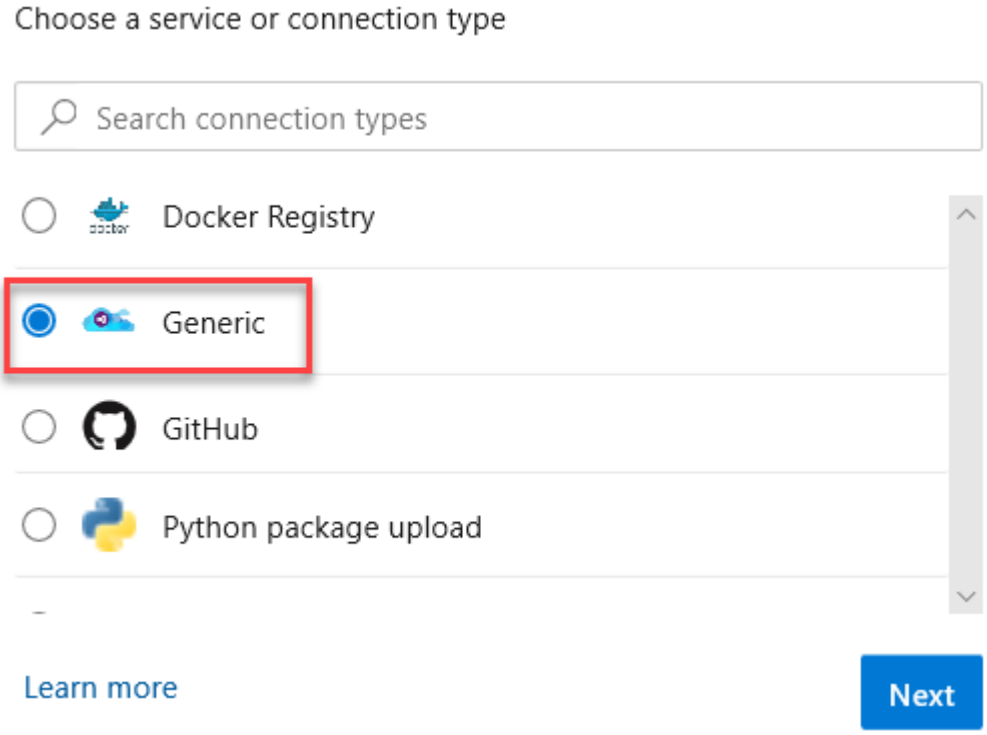


   - Close the **Power Platform Admin** browser window or tab.
6. Create Generic Service Connection
   - Go back to the **Pipeline**.
   - Make sure you still have the **Power Platform Export Solution** task selected.
   - Click **Manage** service sonnection. This will open a new window.

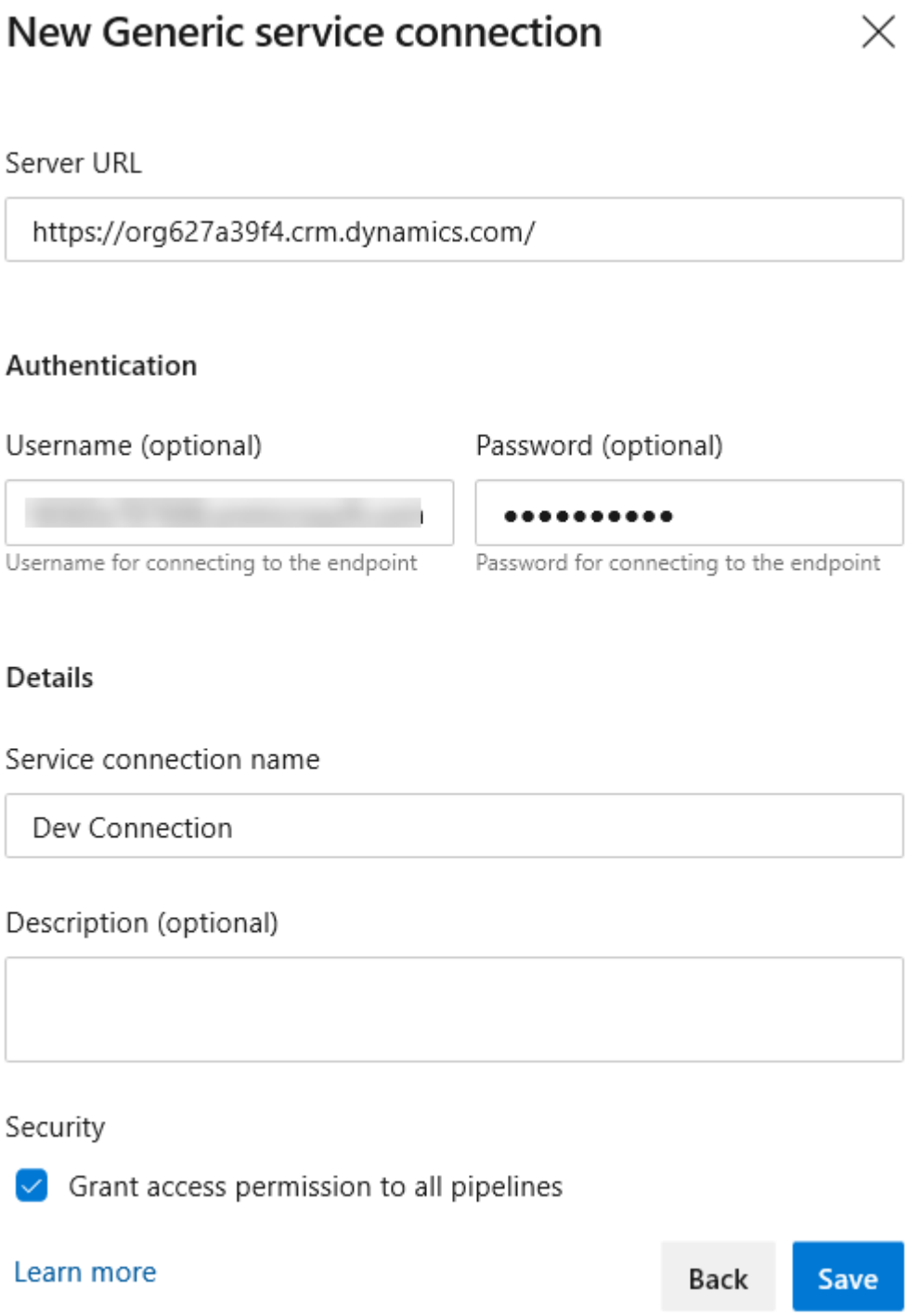
- Click **Create Service Connection**.



- Select **Generic** and click **Next**.

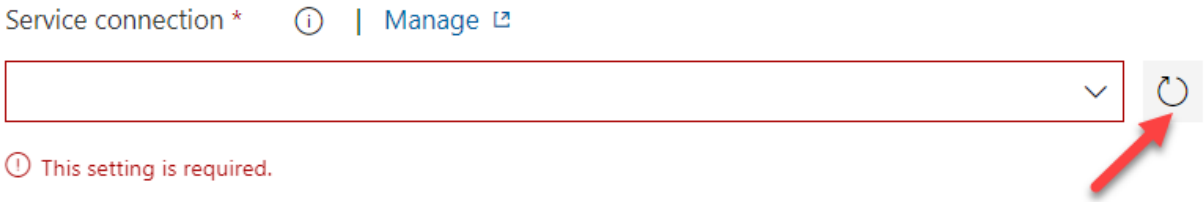


- Paste the **Environment URL** you copied in Server URL, provide your admin credentials, provide a connection name, and click **Save**.

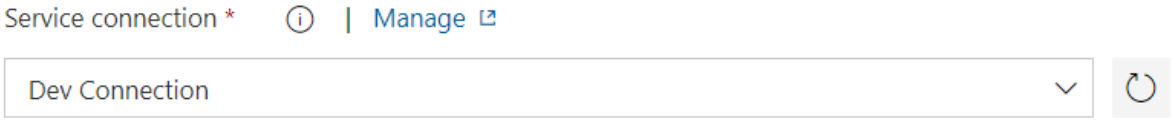


- Close the **Service Connections** browser window or tab.

7. Select the Generic Service Connection you created as the Power Apps Environment URL
   - Go back to the **Build Pipeline** tasks and make sure you still have Power Apps Export Solution task selected.
   - Locate the **Power Apps Environment URL** Column and click **Refresh**.

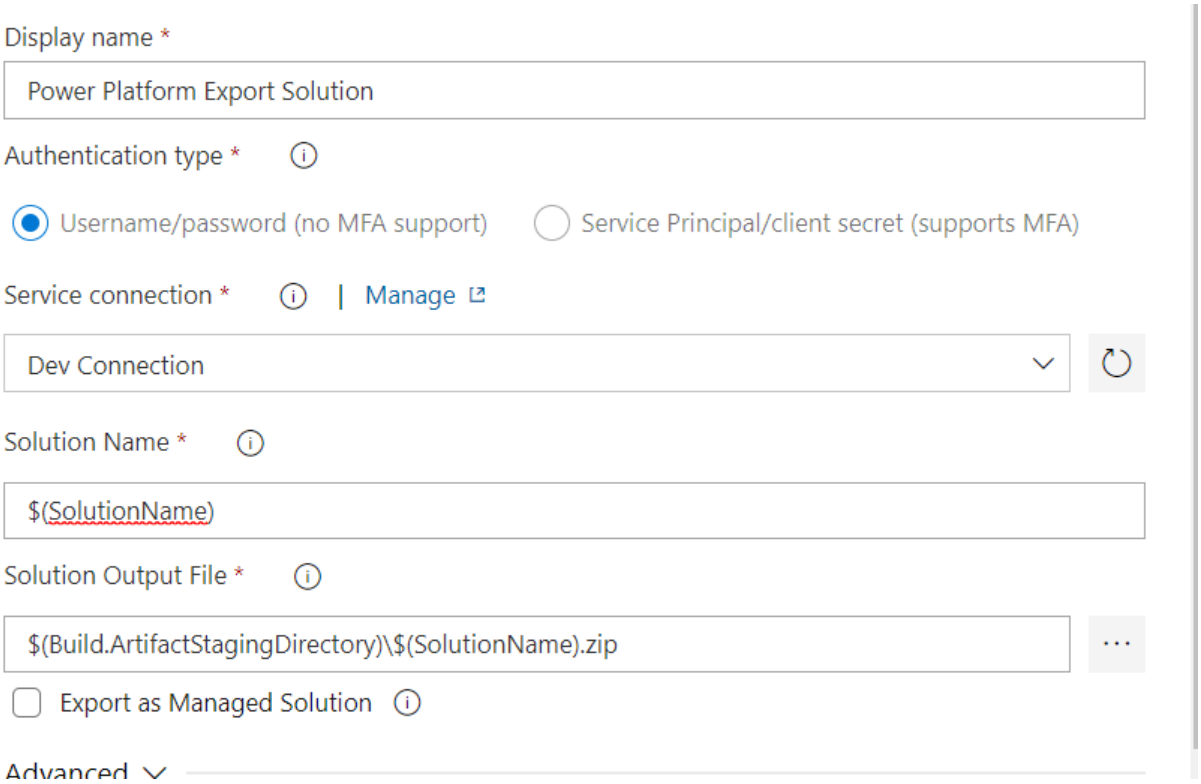


- Select the **Generic Service Connection** you created.

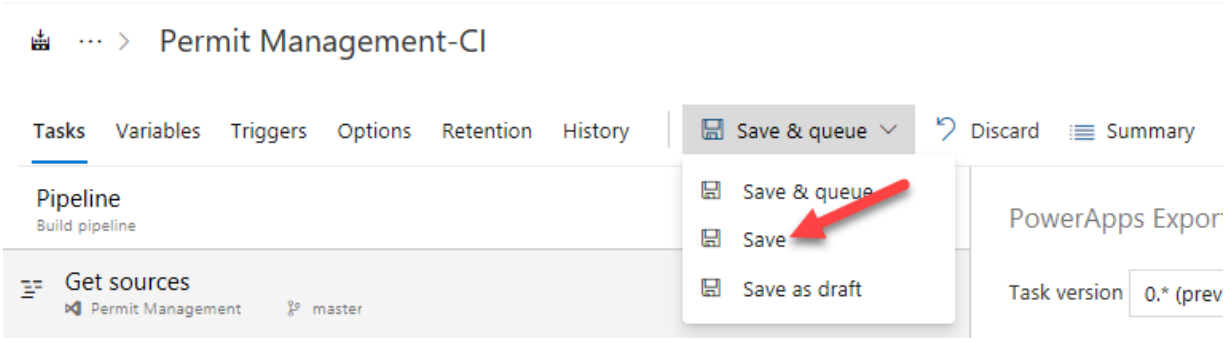


Solution Input File *

- Enter **$(SolutionName)** for **Solution Name**, **$(Build.ArtifactStagingDirectory)$(SolutionName).zip** for **Solution Output File**.



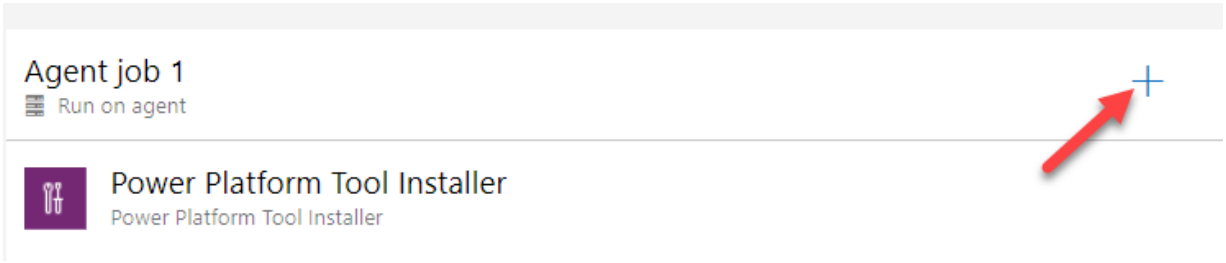
- Click **Save and Queue** and select **Save**.



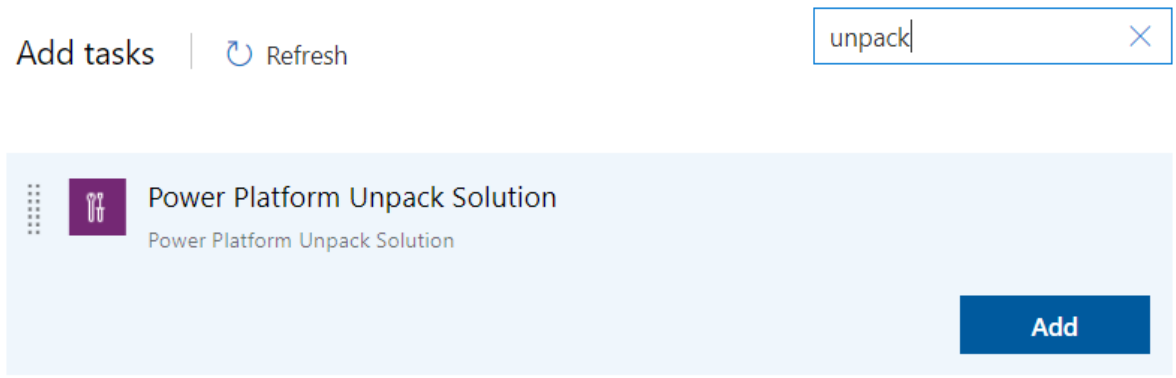
- Click **Save** again.

8. Add Unpack task. This task will take the solution zip file and expand it into a file for each solution component.

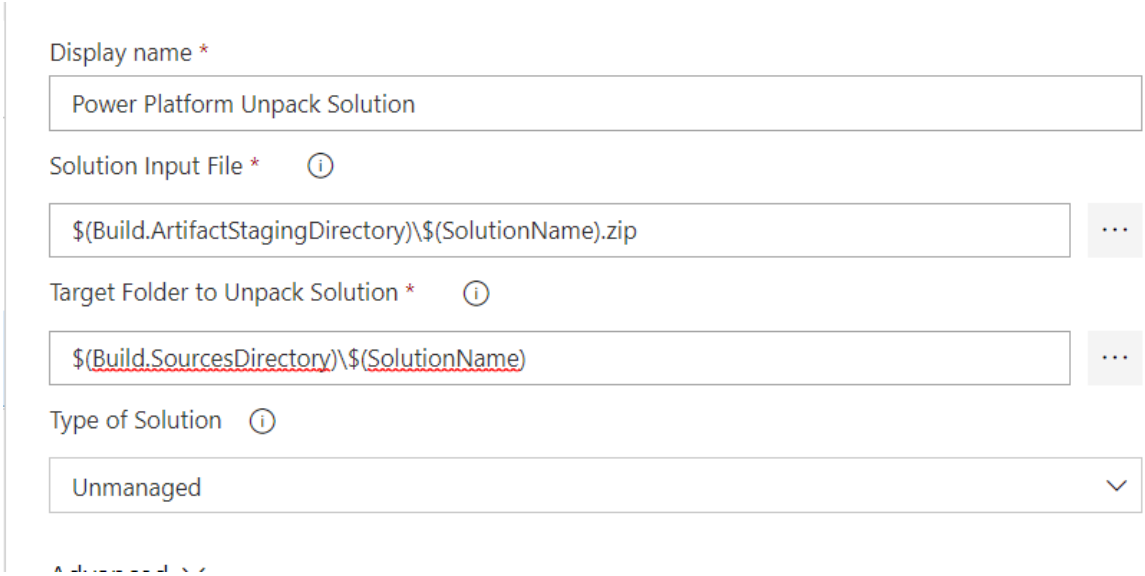
- Click + **Add Task**.



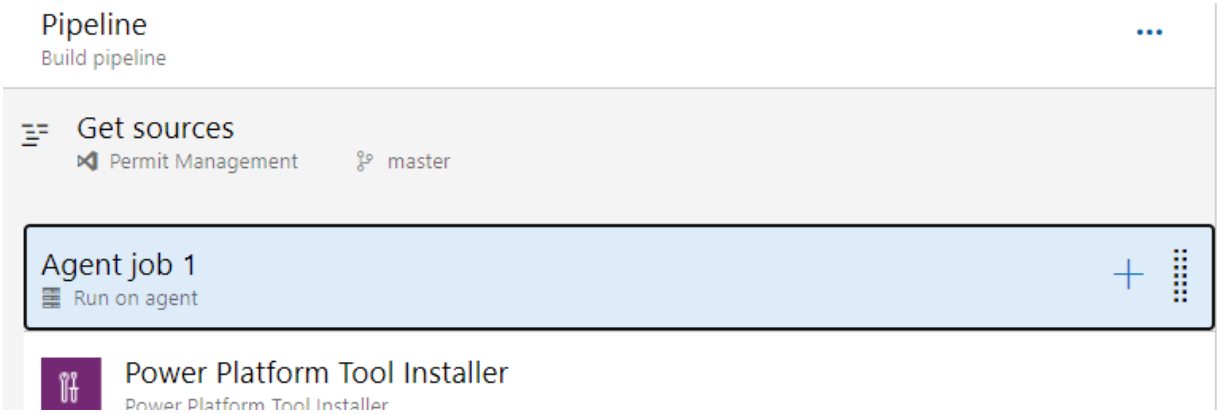
- Search for **Unpack**.
- Hover over **Power Platform Unpack Solution** and click **Add**.



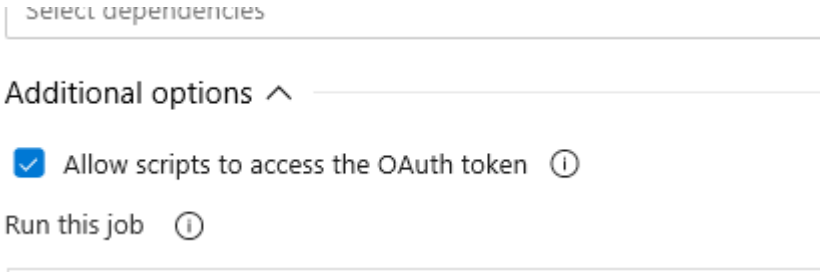
9. Provide Unpack settings information
   - Select the **Unpack** task.
   - Enter **$(Build.ArtifactStagingDirectory)$(SolutionName).zip** for **Solution Input File**, **$(Build.SourcesDirectory)$(SolutionName)** for **Target Folder**.



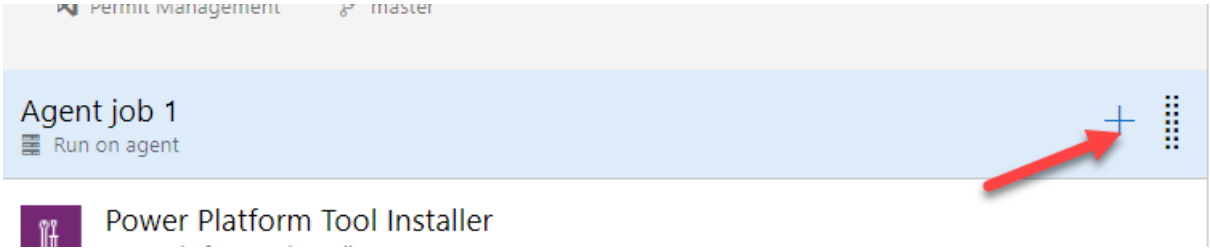
   - Click **Save and Queue** and select **Save**.
   - Click **Save** again.
10. Allow scripts to access the OAuth Token.
    - Select **Agent Job 1**.



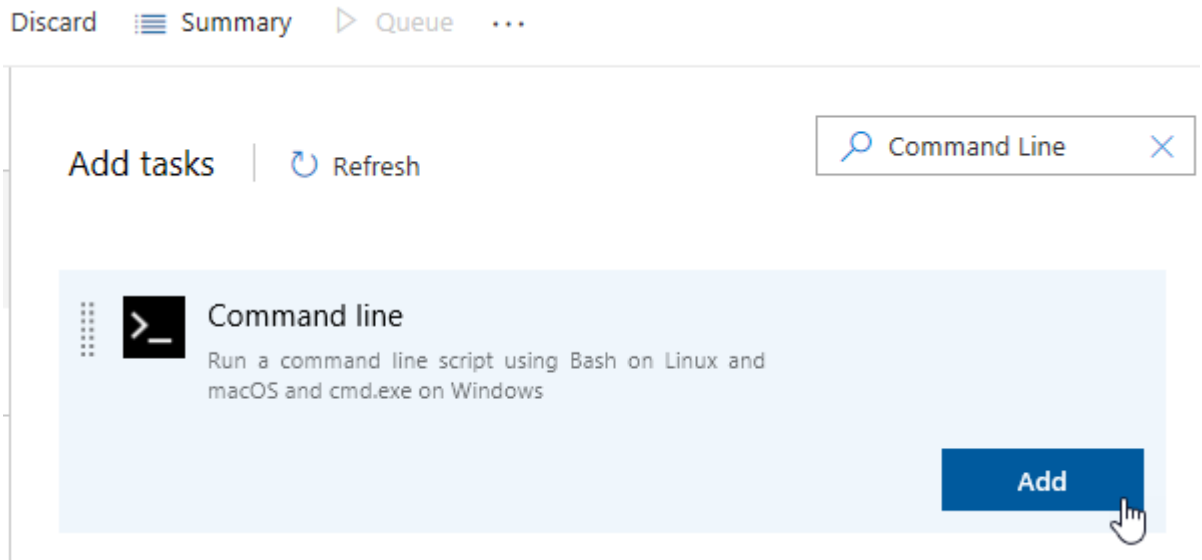
- Scroll down and check the **Allow Scripts to Access the OAuth Token** checkbox.



11. Add Command Line task
    - Click + **Add a Task**.



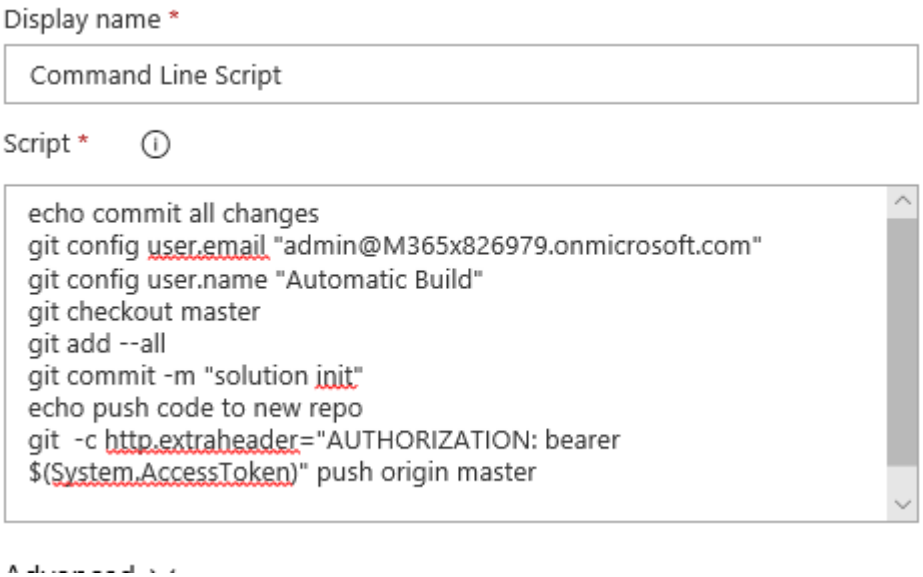
- Search for **Command Line**.
- Hover over **Command Line** and click **Add**.



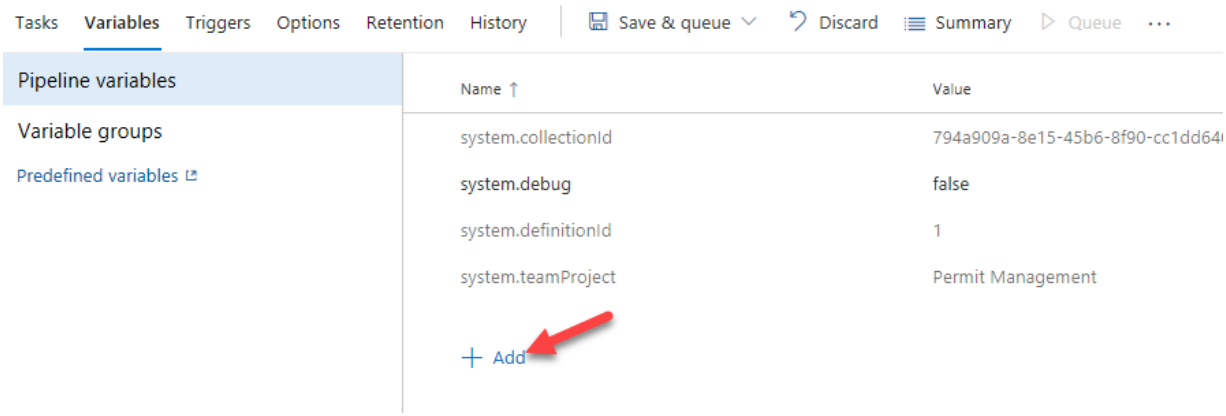
12. Add Scripts to the Command Line task. This task will be used to check in the solution file changes to the repo.
    - Select the **Command Line** task.

- Paste the script below in the **Script** text area. Replace **user@myorg.onmicrosoft.com** with your admin username.

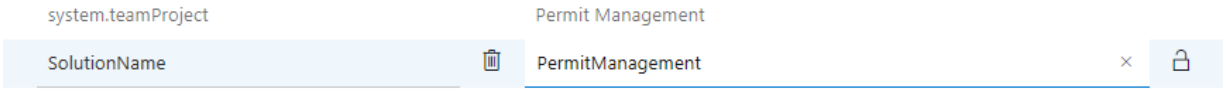
```
echo commit all changes
git config user.email "user@myorg.onmicrosoft.com"
git config user.name "Automatic Build"
git checkout master
git add --all
git commit -m "solution init"
echo push code to new repo
git -c http.extraheader="AUTHORIZATION: bearer $(System.AccessToken)" push origin master
```



13. Add Solution Name variable
    - Select the **Variables** tab.
    - Click + **Add.**



- Enter **SolutionName** for **Name** and **PermitManagement** for **Value**.



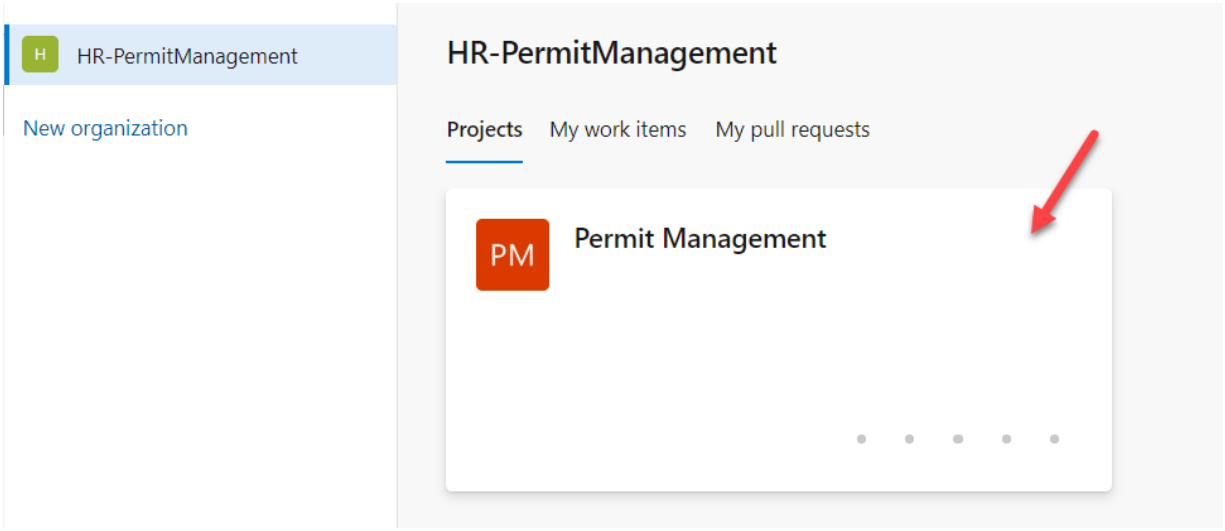
- Click **Save and Queue** and select **Save**.
- Click **Save** again.

52 Exercise #3: Test the Pipeline

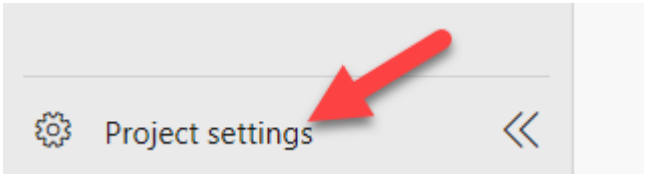
Objective: In this exercise, you will test the build pipeline you created.

52.1 Task #1: Run the Pipeline

1. Open the build pipeline
   - Sign in to Azure DevOps and click to open the **Permit Management** project.

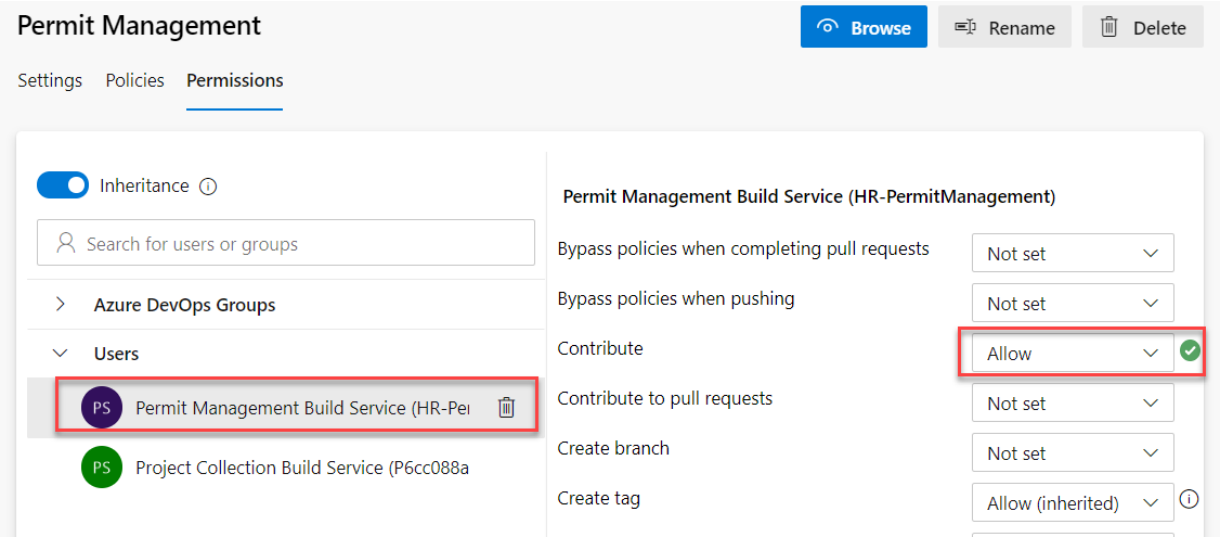


   - Click **Project Settings**.



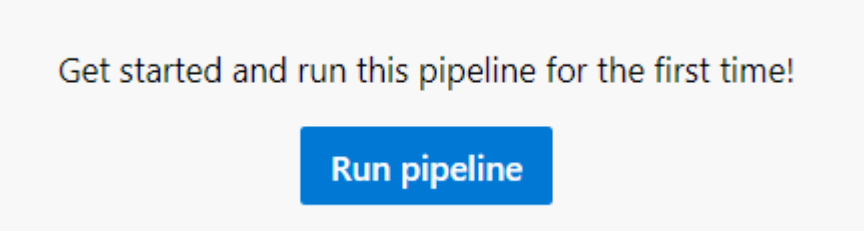
   - Select **Repositories**.

   - Select the **Permit Management** repository.
   - Select the **Permissions** tab.
   - Select **Permit Management Build Service** and **Allow** Contribute.



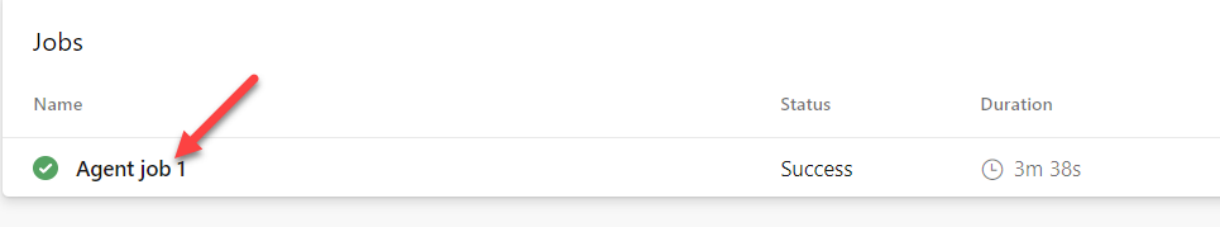
- Select **Pipelines | Pipelines**.

- Select **Permit Management-CI**.
- Click **Run Pipeline**.

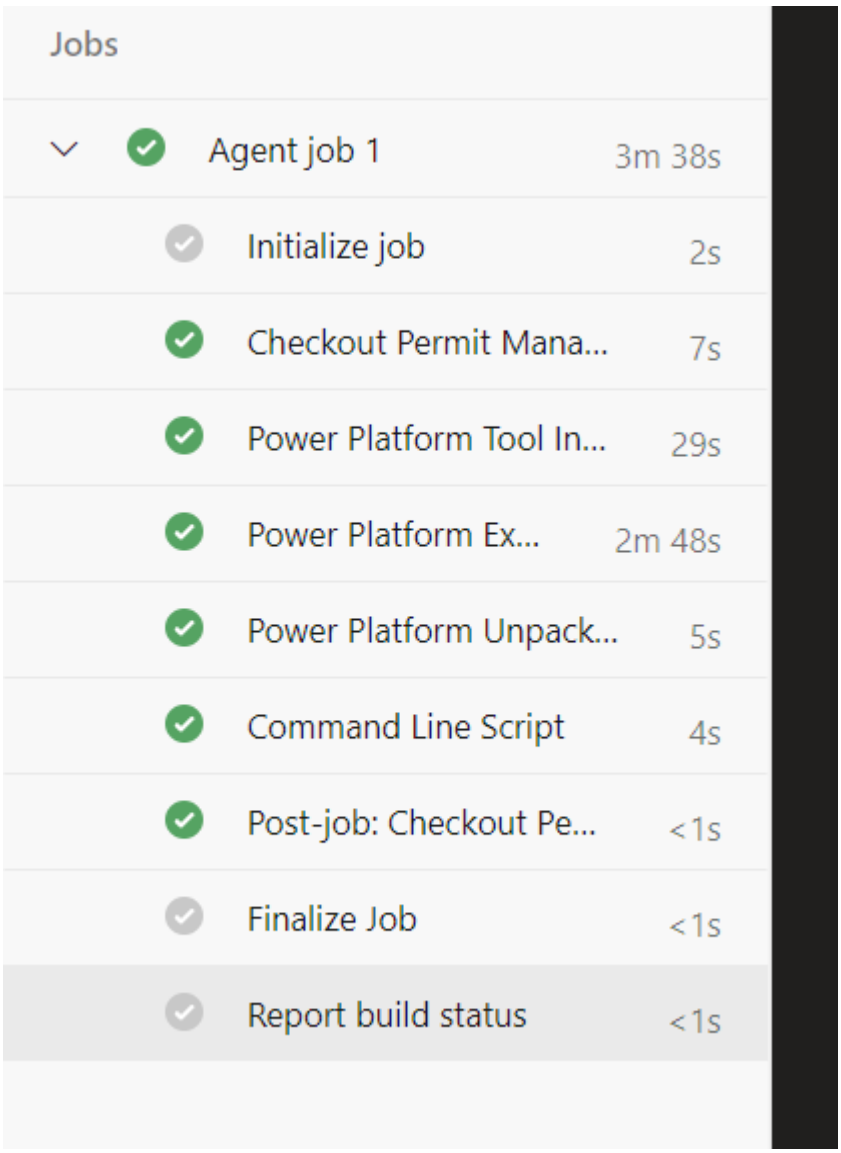


- Click **Run** again and wait.

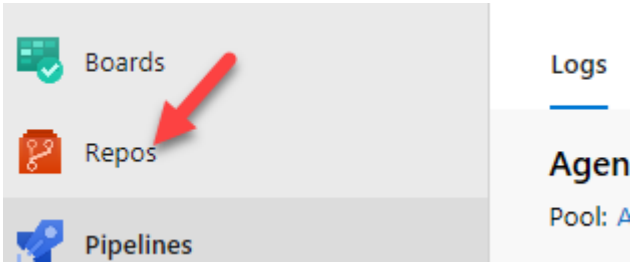
- Wait until the job completed and click to open it.



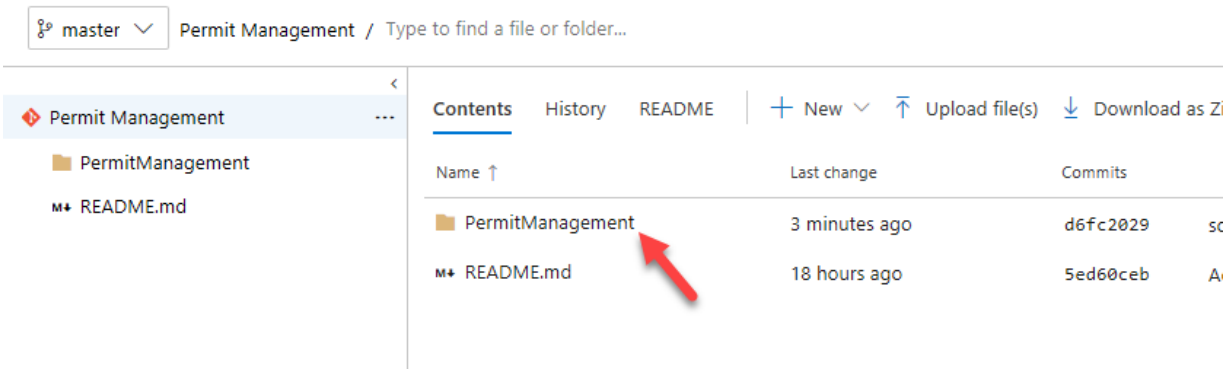
- The Build tasks should run and succeed



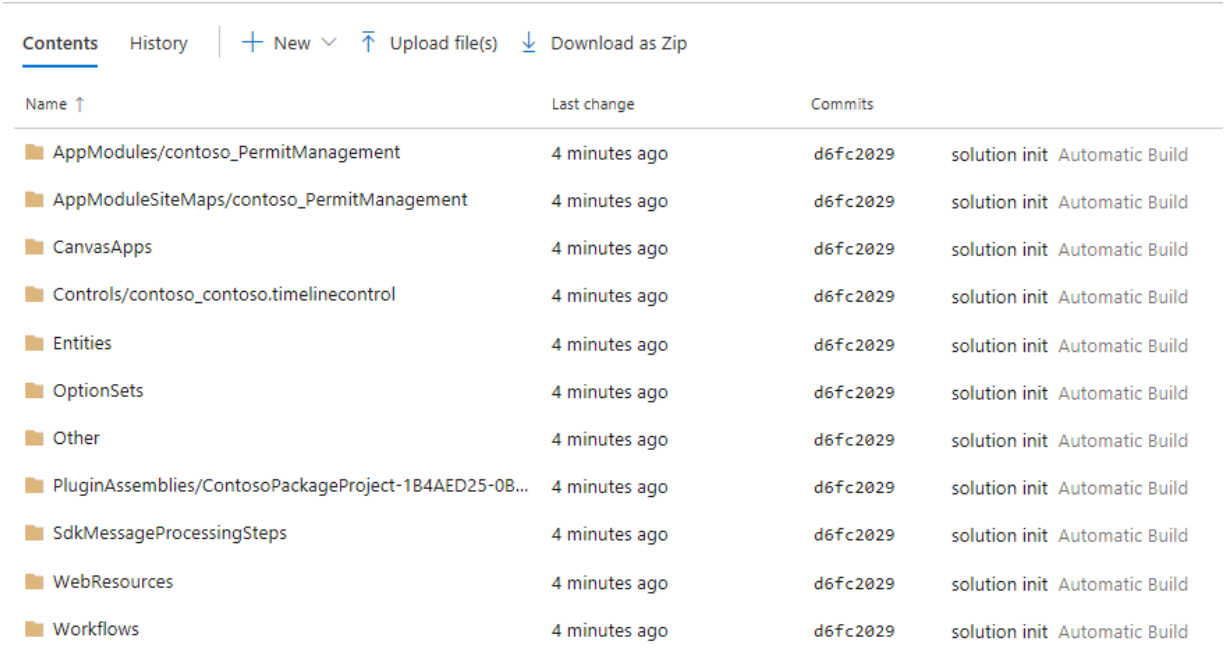
2. Review the Repository
   - Select Repos.



   - You should see **PermitManagement** folder. Click to open the folder.



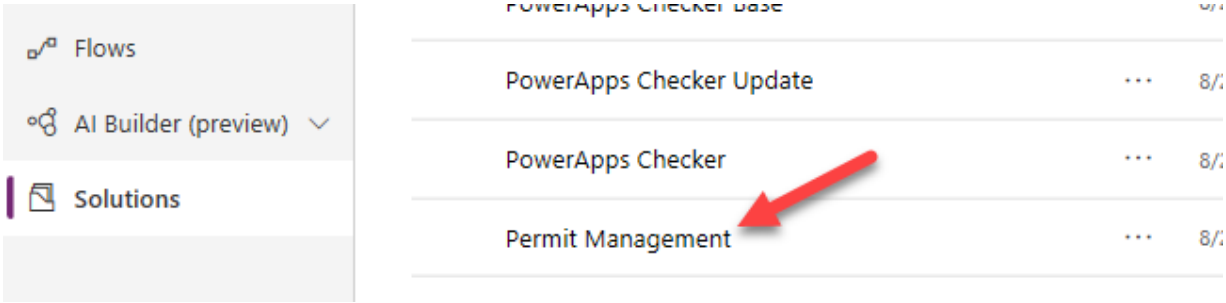
- The content of the folder should look like the image below.



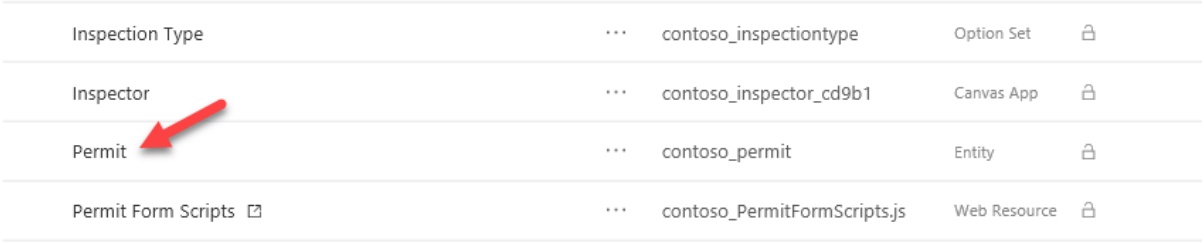
You may examine the content of each folder.

52.2 Task #2: Modify Solution

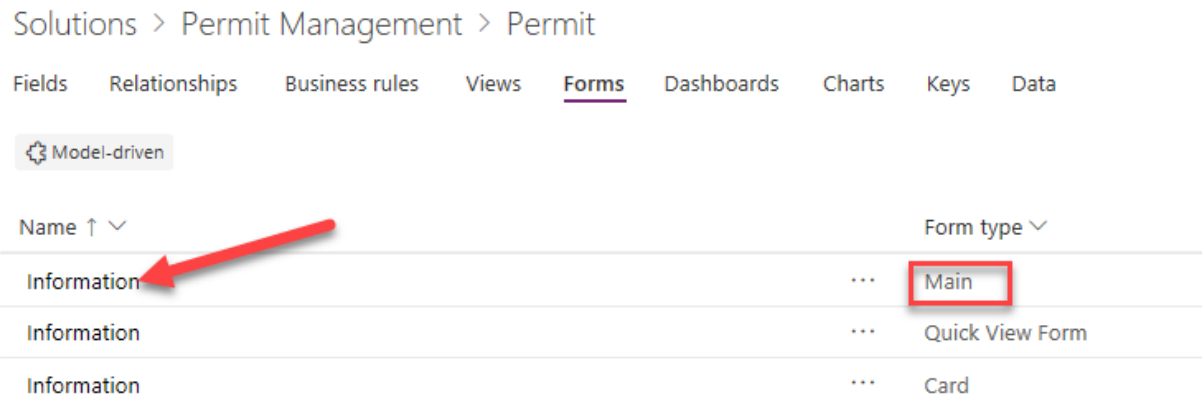
1. Open the Permit Management solution
   - Sign in to Power apps maker portal and make sure you have the **Dev** environment selected.
   - Select **Solutions**.
   - Click to open the **Permit Management** solution.



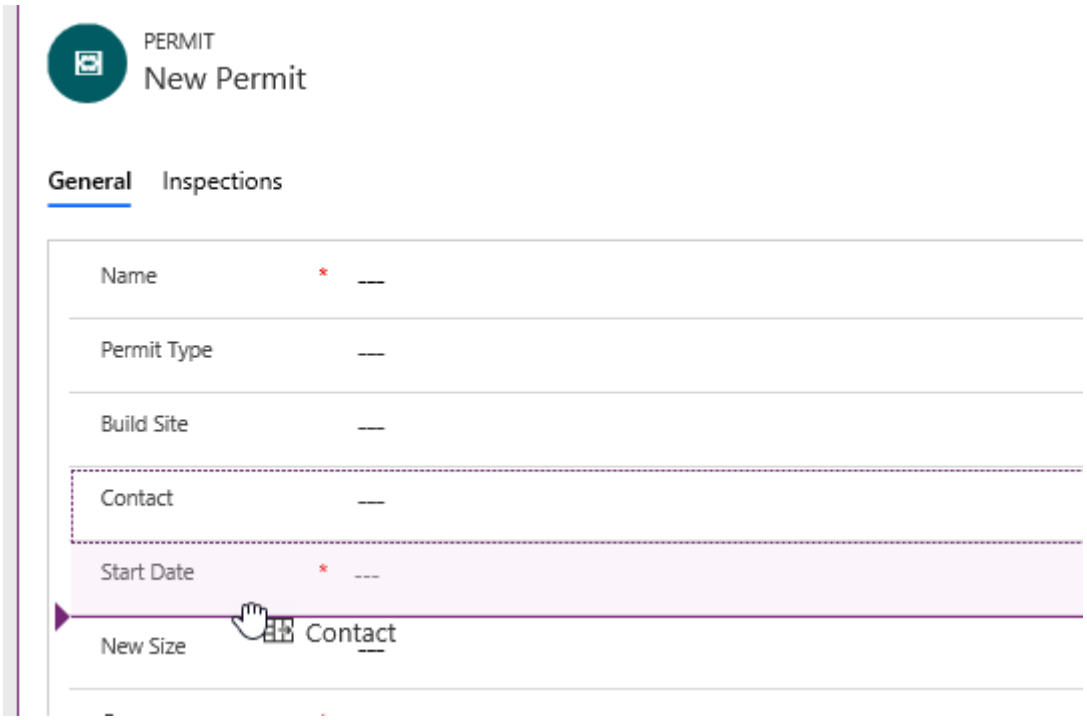
2. Open the Permit Table form for edit
   - Click to open **Permit** Table.



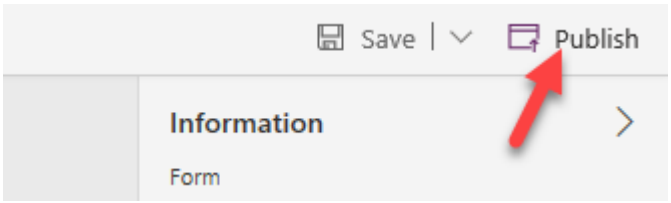
- Select the **Forms** tab and click to open the **Main** form.



3. Move the Contact lookup Column
   - Drag the **Contact** lookup Column and drop it between the **Start Date** and **New Size** Columns.

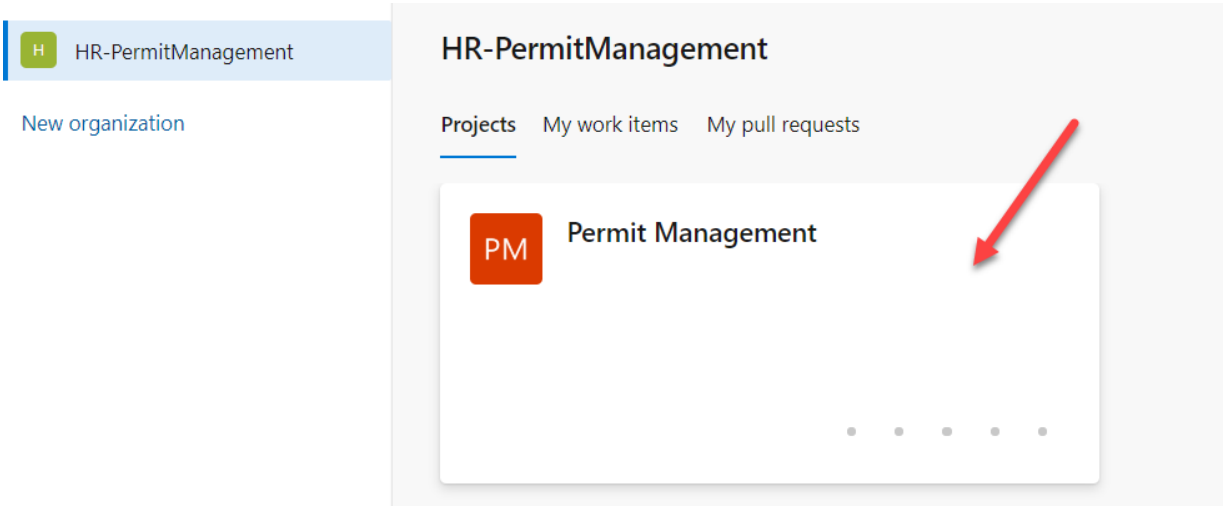


   - Click **Save**.
   - Click **Publish** and wait for the publishing to complete.

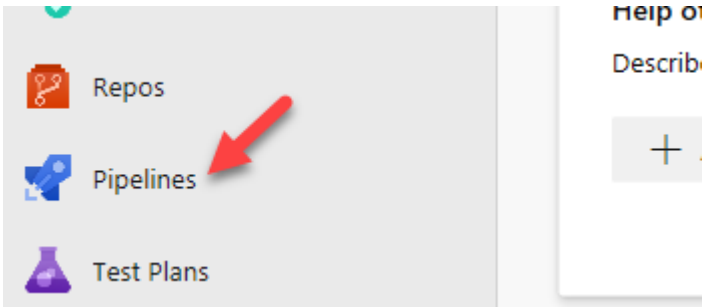


52.3 Task #3: Run Build Pipeline

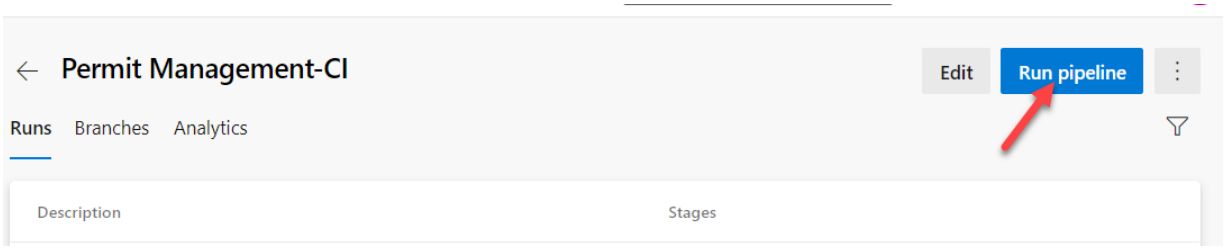
1. Open Permit Management DevOps project
   - Sign in to Azure DevOps
   - Click to open the **Permit Management** project



2. Run the build pipeline again
   - Select **Pipelines**.

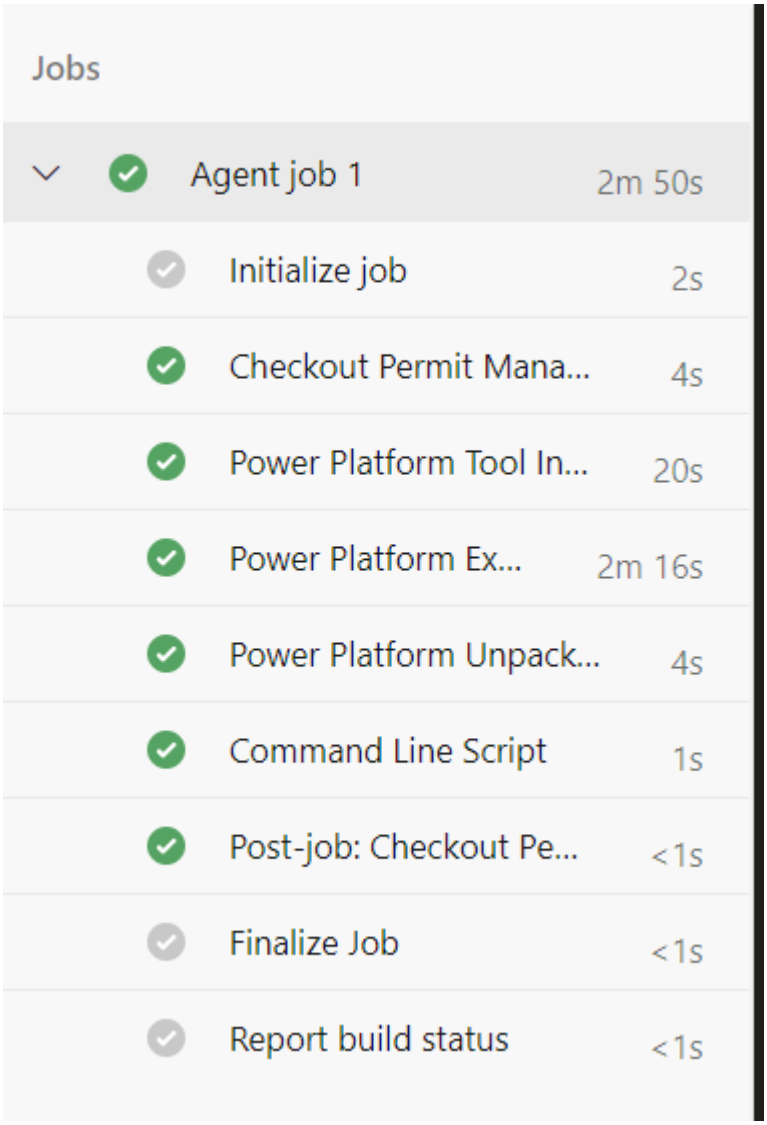


   - Select **Permit Management-CI**.
   - Click **Run Pipeline**.

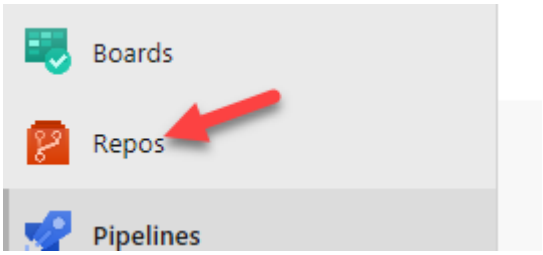


   - Click **Run** and wait for the run to complete.
   - Open the job after it completes.

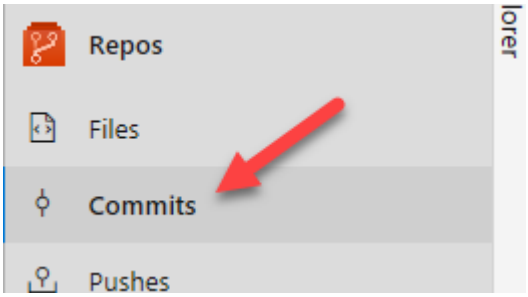
   - All the tasks should succeed again.



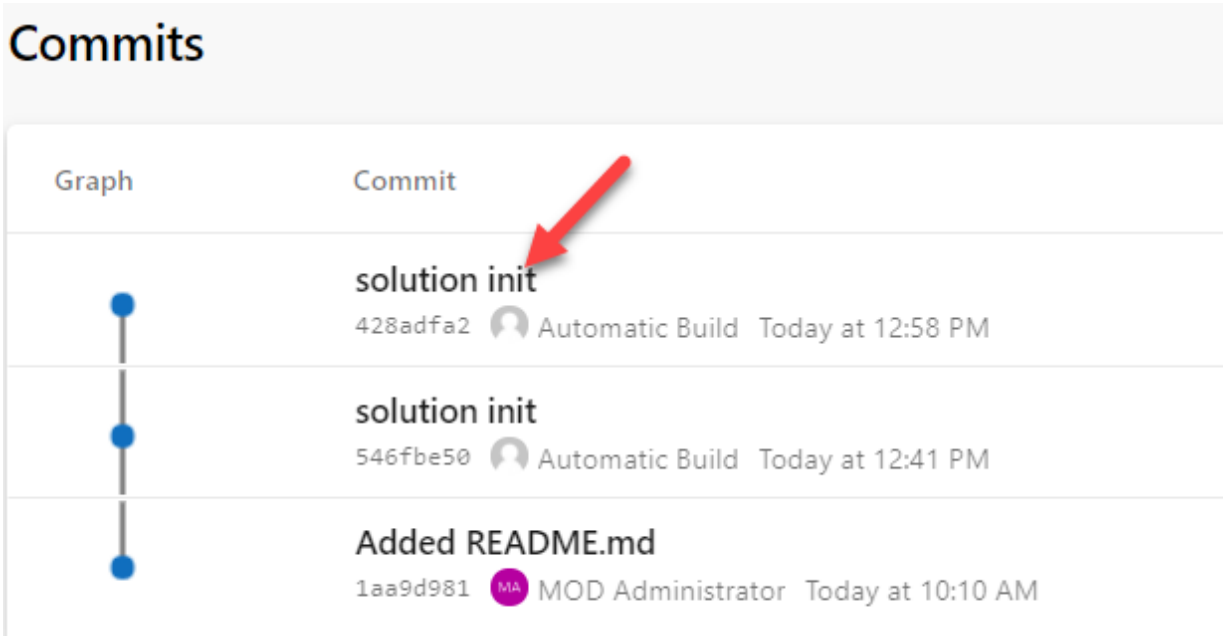
3. Check the Repository for the new changes
   - Select **Repos**.



   - Select **Commits**.

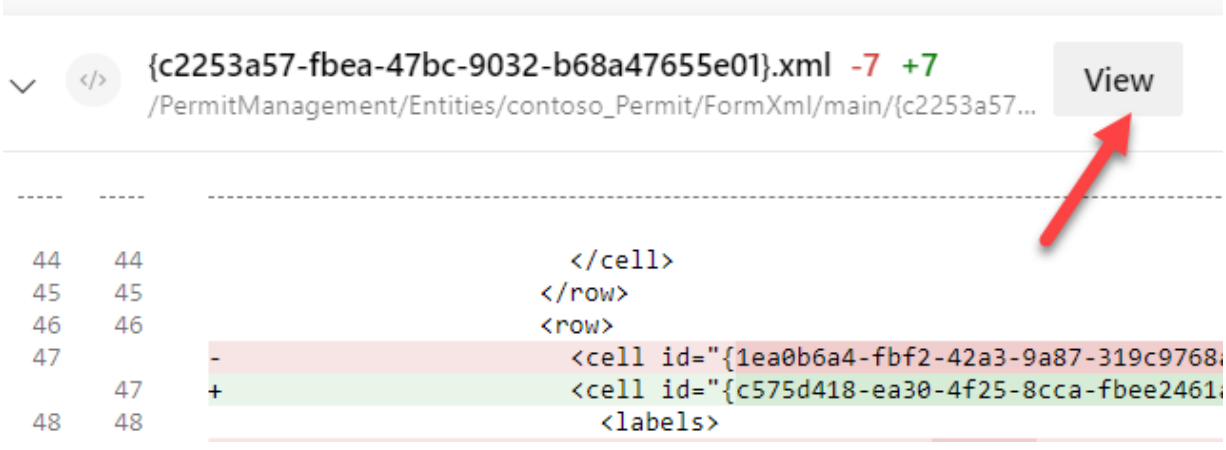


   - Click to open then topmost commit.

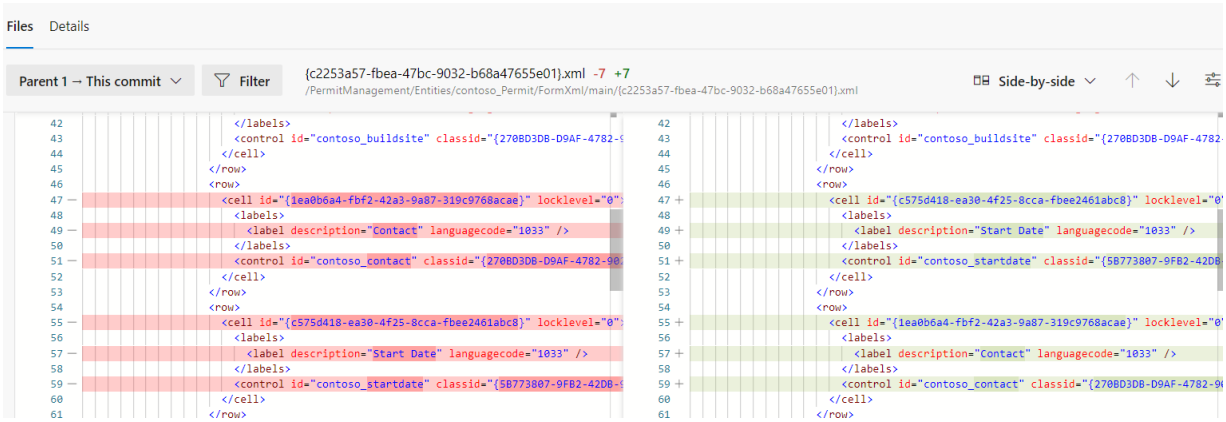


- The changeset should look like the image below.

4. View side-by-side.
   - Click **View**.



   - Side-by-side view should load.


Files
Details
Parent 1 → This commit
Filter
{c2253a57-fbea-47bc-9032-b68a47655e01}.xml -7 +7
/PermitManagement/Entities/contoso_Permit/FormXml/main/{c2253a57-fbea-47bc-9032-b68a47655e01}.xml
Side-by-side