

# Custom Connector Lab

Power Platform Workshop

# Step 1. Create a new Function using the Azure Functions Extension.

Select the [Azure Functions extension for Visual Studio Code](#) and add a new function using the lightning bolt icon.

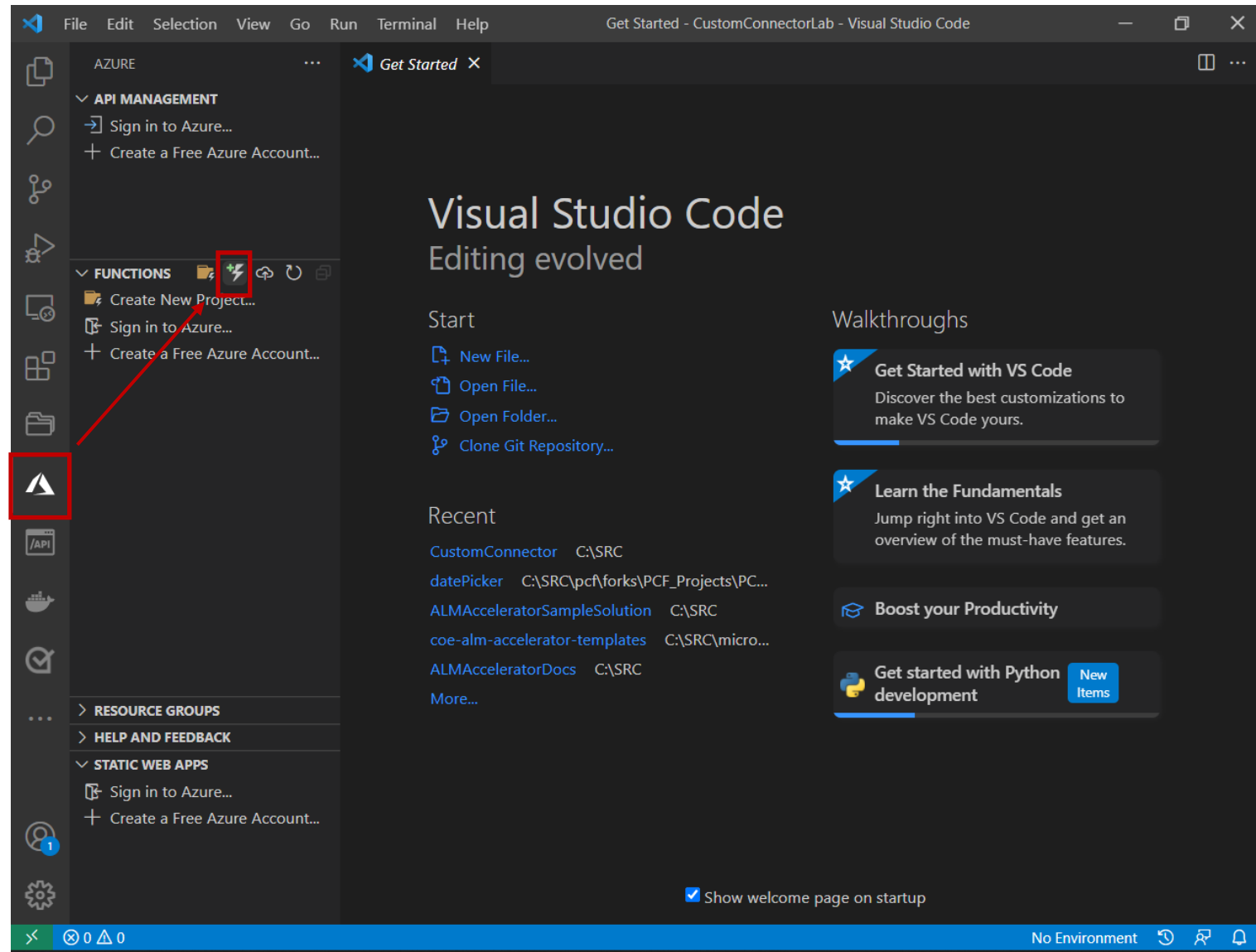
Click yes if prompted to create a function project

Create your function project with the following properties

- Language: C#
- .NET runtime: .NET core 3
- Template: HTTP trigger with OpenAPI
- Name: GetWeather
- Namespace: CompanyName.Weather
- AccessRights: Anonymous\*

\*For the purpose of this lab we will use anonymous access rights.

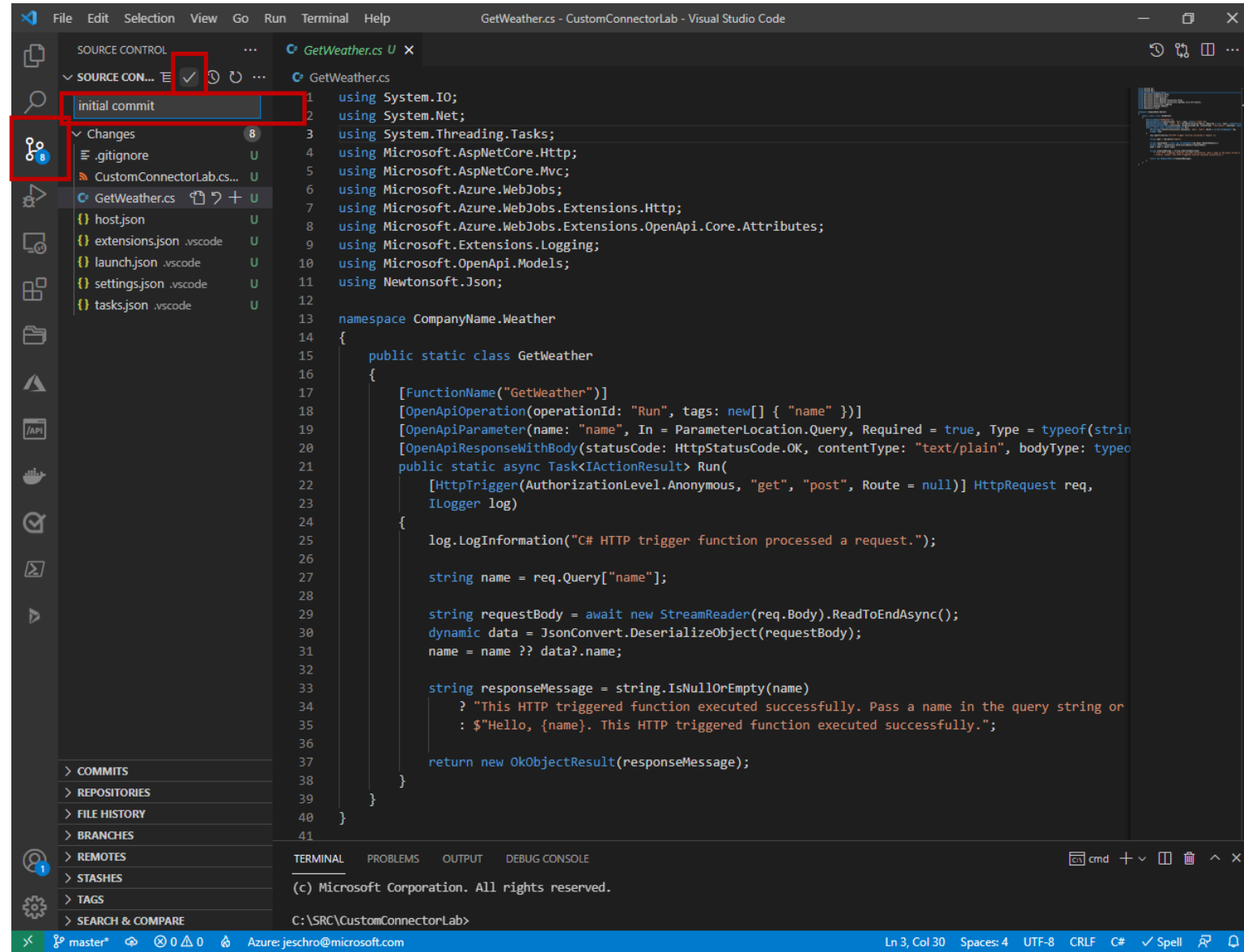
However, it is recommended to always secure your functions



# Step 2. Check in the Function to version control.

Select the [GitLens extension](#)

Enter a commit message and click the commit (✓) button

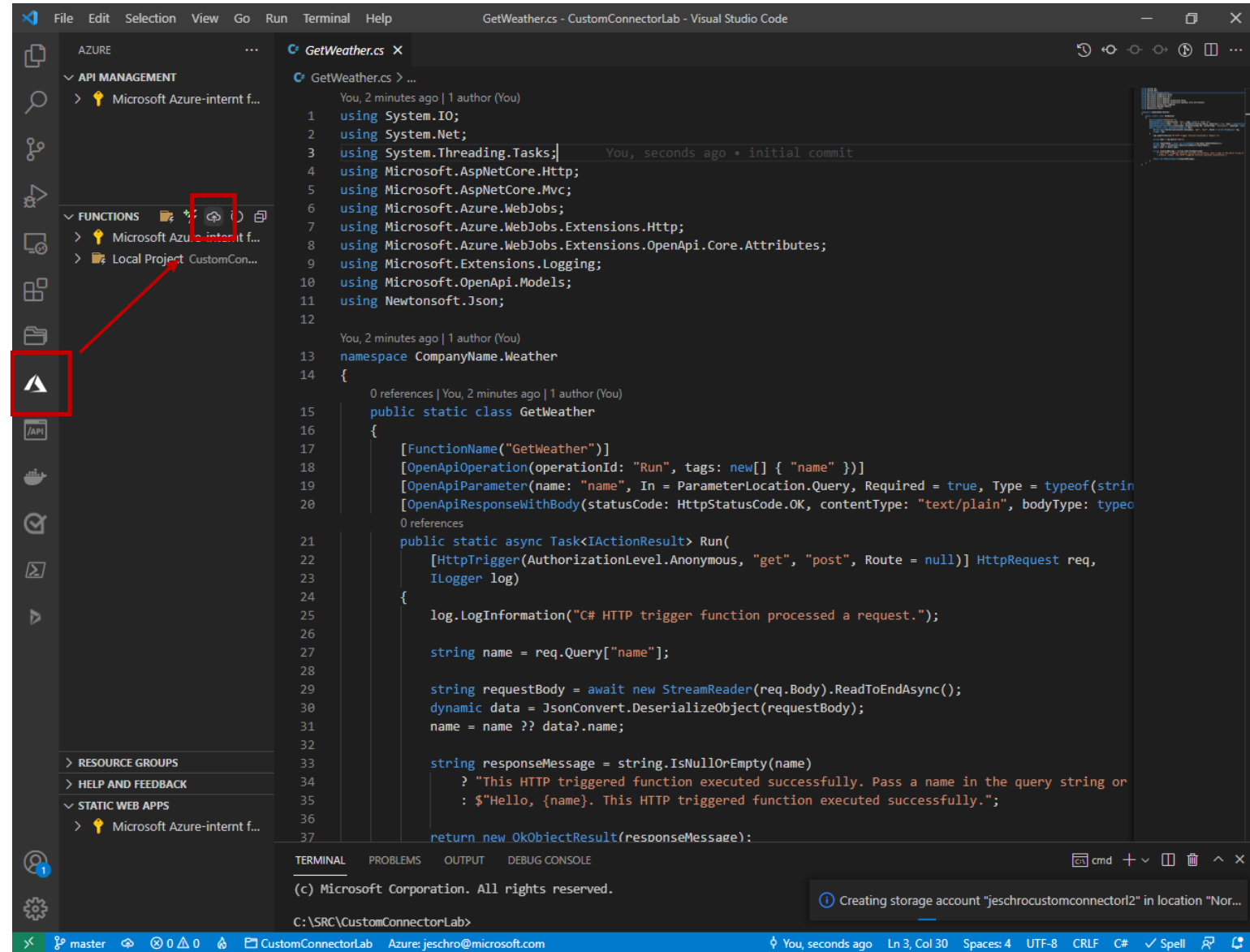


# Step 3. Deploy this function into the Azure Subscription.

Select the [Azure Functions extension for Visual Studio Code](#) and deploy your function app to Azure by clicking the cloud icon.

Create new Function App in Azure with the following properties

- Name: ?(must be globally unique)
- Runtime: .NET core 3.1
- Location: North Europe (recommended)



# Step 4. Run the Function interactively in a browser.

In this example the function was deployed to a function with the name "GetTemperature" therefore can be called using the following URL

<https://jeschro-customconnectorlab.azurewebsites.net/api/GetWeather>

This returned the following:

*"This HTTP triggered function executed successfully. Pass a name in the query string or in the request body for a personalized response."*

If the query string parameter **name** is passed in like the following:

<https://jeschro-customconnectorlab.azurewebsites.net/api/GetWeather?name=Jens>

Will return the following:

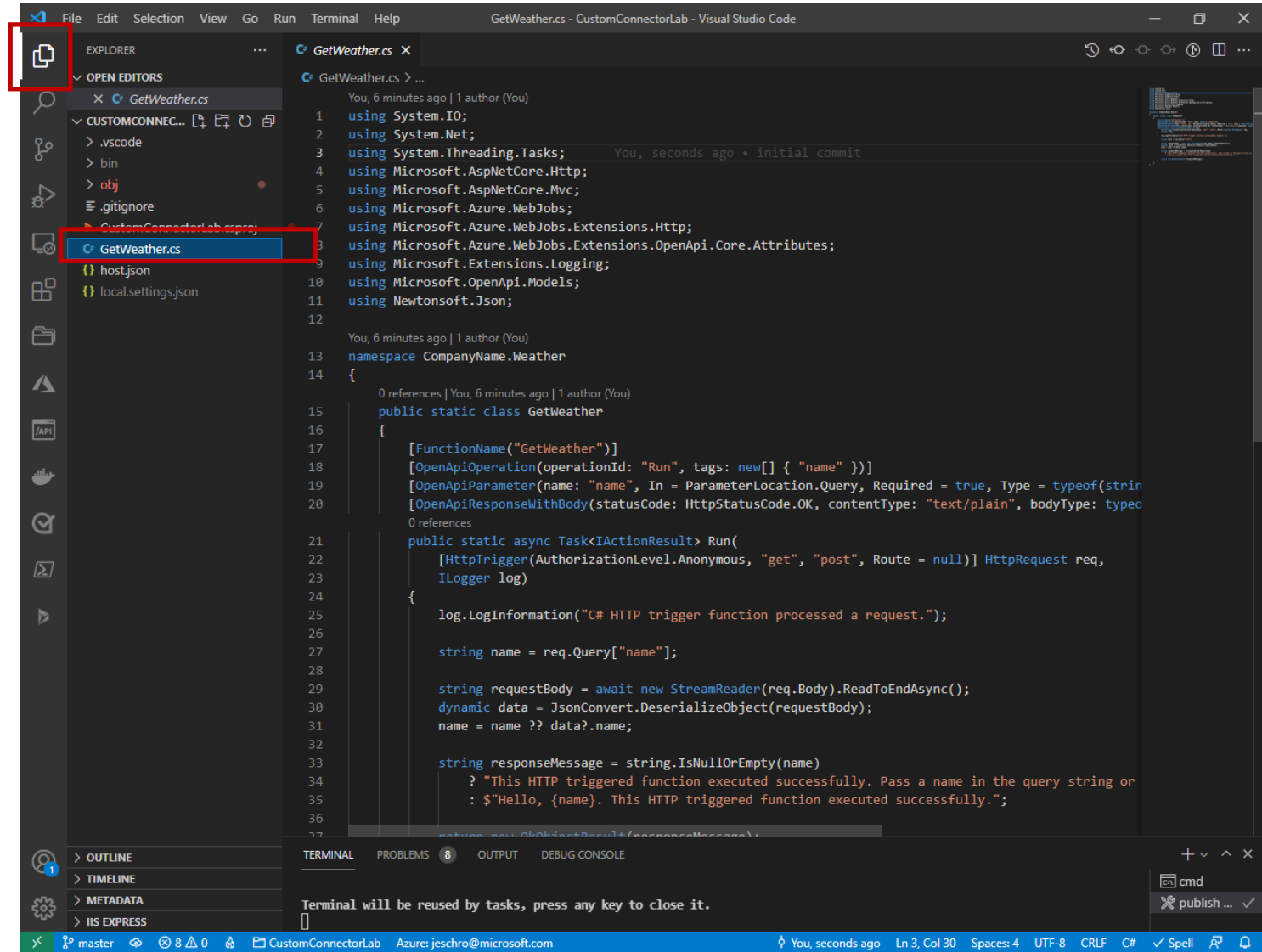
*"Hello, Jens. This HTTP triggered function executed successfully."*

While [Azure Functions extension for Visual Studio Code](#) makes it easy to create an HTTP based function the default template returns a string, it does NOT return JSON...So this functions cannot be called from a Flow in Power Automate or Power Apps. Both requires connectors to return JSON

We will update the function to return JSON in the next steps

# Step 5. Open the function implementation.

Click on the files view of Visual Studio code and the GetWeather.cs function file



## Step 6. Add the System.Collections.Generic namespace

Add the following line of code at the end of the existing using statements in the top of your function code

We will be implementing logic using the List type in the System.Collections.Generic namespace in the following steps

```
12      using System.Collections.Generic;
```

# Step 7. Add supporting classes

Add the weatherForecast and location classes to the Function App

Insert the code to the right inside the namespace but outside the static function class (before the last } )

```
56     public class weatherForecast {  
57         public List<location> locations { get; set; }  
58     }  
59  
60     public class location {  
61         public string locationName { get; set; }  
62         public string forecast { get; set; }  
63     }
```



# Step 8. Edit the function to weather forecast

Replace the following code (lines 33-37) :

```
34         string responseMessage = string.IsNullOrEmpty(name)
35             ? "This HTTP triggered function executed successfully. Pass a name in the query string or in the
request body for a personalized response."
36             : $"Hello, {name}. This HTTP triggered function executed successfully.";
37
38         return new OkObjectResult(responseMessage);
```

With this code :

```
34         // Weather Forecast Object
35         weatherForecast weatherForecastObject = new weatherForecast();
36         weatherForecastObject.locations = new List<location> {
37             new location { locationName = "Bergen", forecast = "Rain" },
38             new location { locationName = "Oslo", forecast = "Cloudy" },
39             new location { locationName = "Stavanger", forecast = "Sunny" }
40         };
41
42         // Find location matching querystring parameter name
43         var forecast = weatherForecastObject.locations.Find(x => x.locationName.ToLower() == name.ToLower());
44
45         // if location matching querystring parameter name is not found return error message
46         if(forecast == null) {
47             forecast = new location { locationName = name, forecast = $"Location '{name}' not found" };
48         }
49         // return weather forecast for location
50         return new OkObjectResult(jsonForecast);
```

# Step 9. Edit the function to return JSON continued

You code should look like the code in the following pages

# Full code page 1 of 2

```
using System.IO;
using System.Net;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Azure.WebJobs;
using Microsoft.Azure.WebJobs.Extensions.Http;
using Microsoft.Azure.WebJobs.Extensions.OpenApi.Core.Attributes;
using Microsoft.Extensions.Logging;
using Microsoft.OpenApi.Models;
using Newtonsoft.Json;
using System.Collections.Generic;

namespace Equinor.CustomConn
{
    public static class HttpTrigger
    {
        [FunctionName("HttpTrigger")]
        [OpenApiOperation(operationId: "Run", tags: new[] { "name" })]
        [OpenApiParameter(name: "name", In = ParameterLocation.Query, Required = true, Type = typeof(string), Description = "The **Name** parameter")]
        [OpenApiResponseWithBody(statusCode: HttpStatusCode.OK, contentType: "text/plain", bodyType: typeof(string), Description = "The OK response")]
        public static async Task<IActionResult> Run(
            [HttpTrigger(AuthorizationLevel.Anonymous, "get", "post", Route = null)] HttpRequest req,
            ILogger log)
        {
            log.LogInformation("C# HTTP trigger function processed a request.");

            string name = req.Query["name"];

            string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
            dynamic data = JsonConvert.DeserializeObject(requestBody);
            name = name ?? data?.name;
        }
    }
}
```

# Full code page 2 of 2

```
// Weather Forecast Object
weatherForecast weatherForecastObject = new weatherForecast();
weatherForecastObject.locations = new List<location> {
    new location { locationName = "Bergen", forecast = "Rain" },
    new location { locationName = "Oslo", forecast = "Cloudy" },
    new location { locationName = "Stavanger", forecast = "Sunny" }
};

// Find location matching querystring parameter name
var forecast = weatherForecastObject.locations.Find(x => x.locationName.ToLower() == name.ToLower());

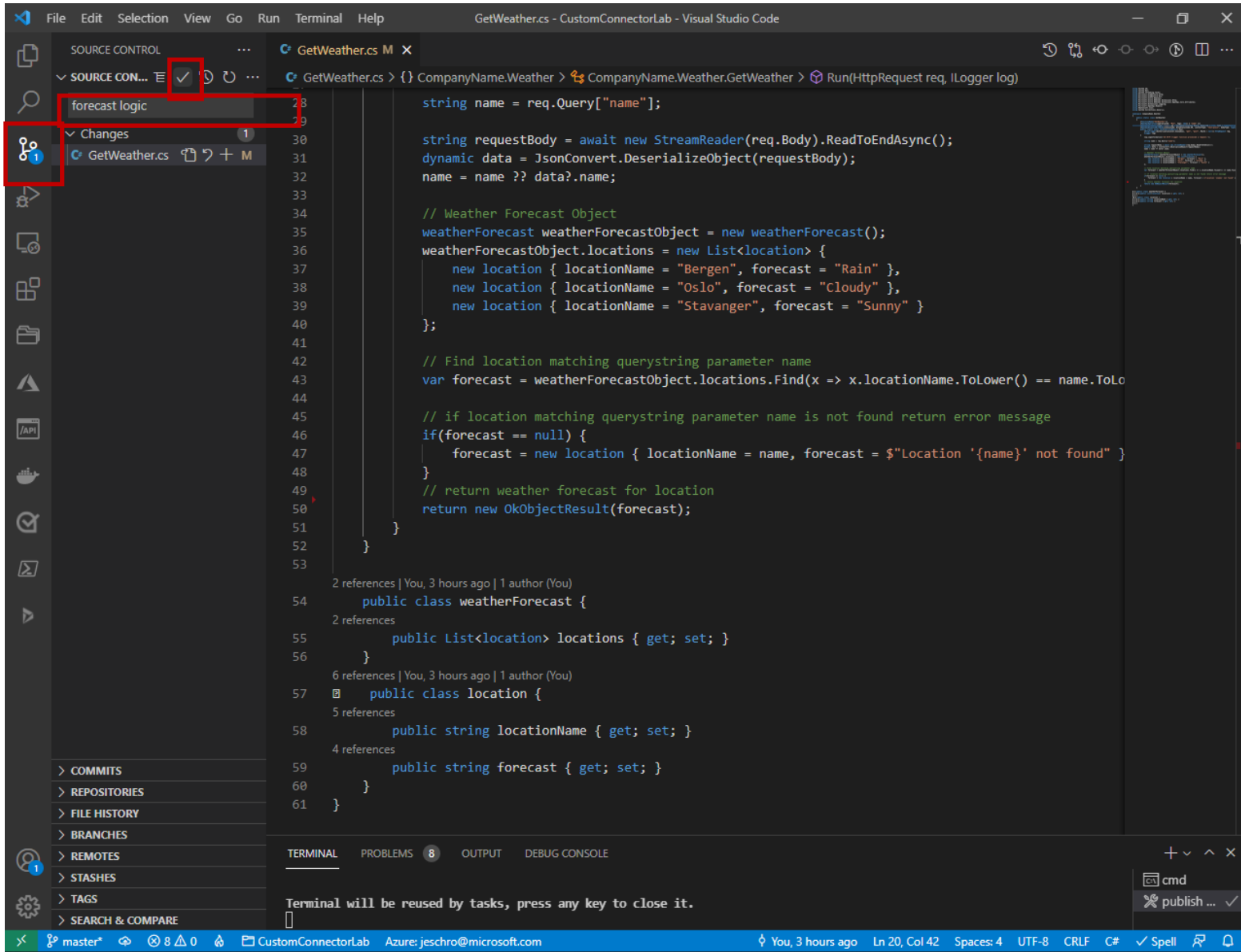
// if location matching querystring parameter name is not found return error message
if(forecast == null) {
    forecast = new location { locationName = name, forecast = $"Location '{name}' not found" };
}
// return weather forecast for location
return new OkObjectResult(forecast);
}
}

public class weatherForecast {
    public List<location> locations { get; set; }
}

public class location {
    public string locationName { get; set; }
    public string forecast { get; set; }
}
}
```

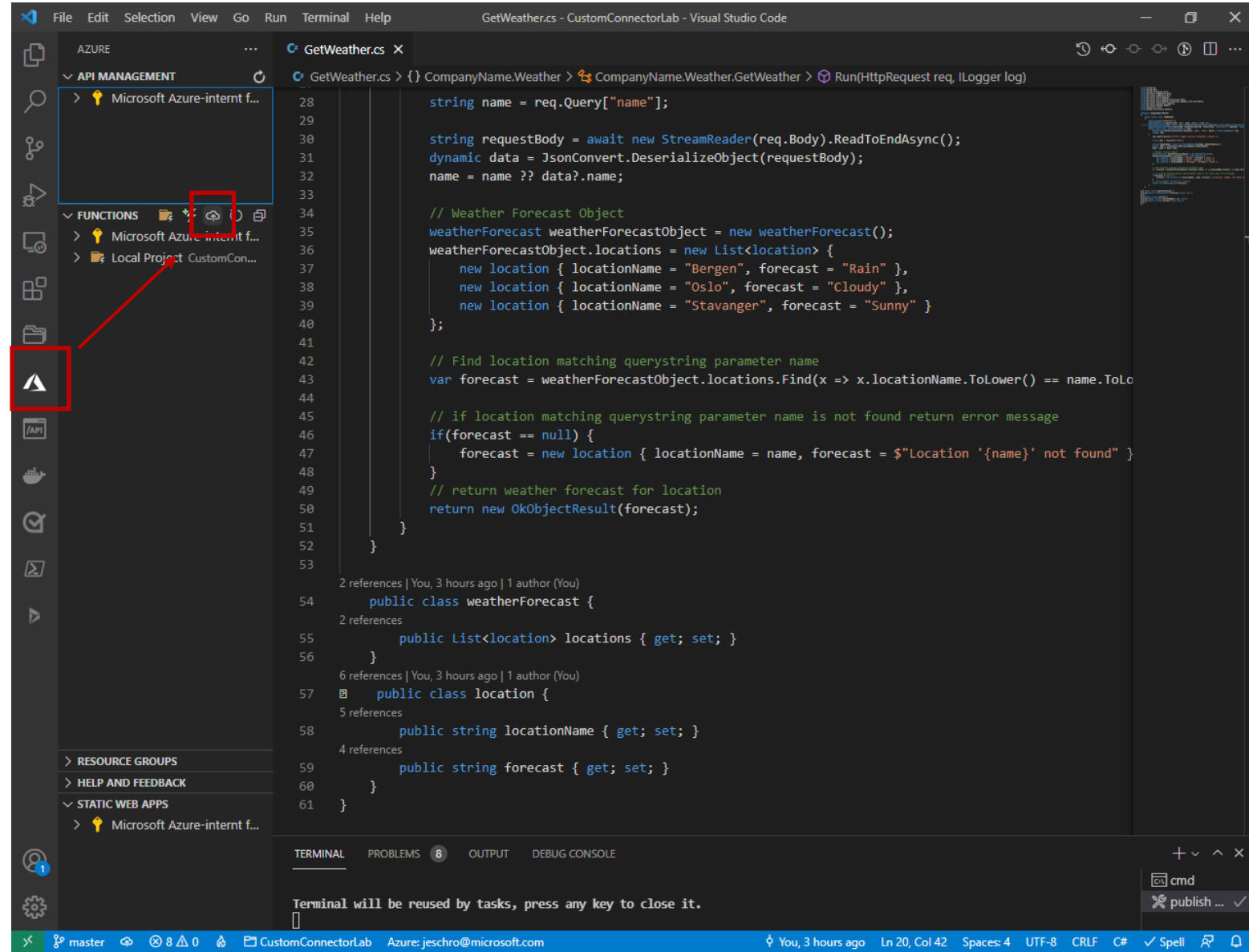
# Step 10. Check in this update.

Save and commit your changes



# Step 11. Deploy these updates:

Deploy your changes to the Azure Function App you created earlier.

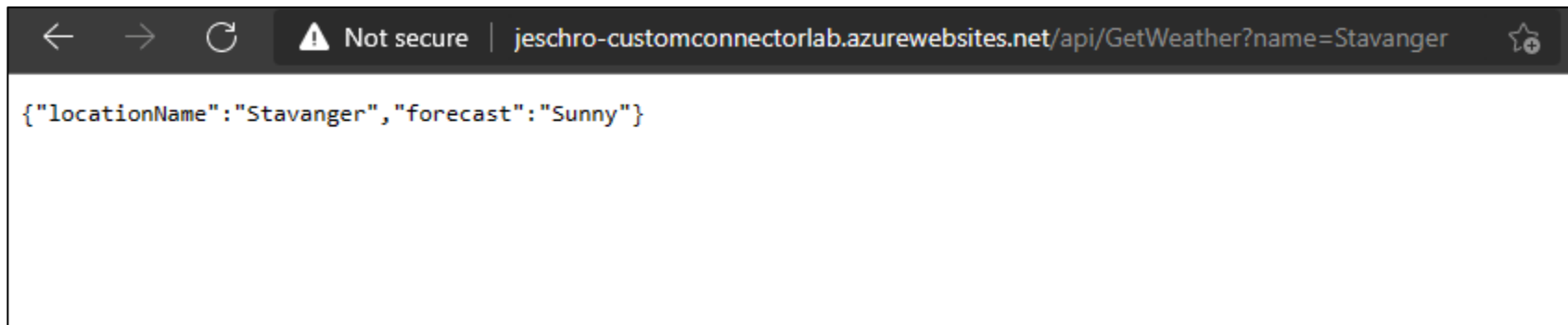


# Step 12. Run the function interactively from the Browser.

Running the updated browser will now return the following JSON

Note, the name parameter is required for the function to work properly. We will handle this in the Custom Connector implementation

***Congratulations your Azure function is now ready to be called by Power Apps!***

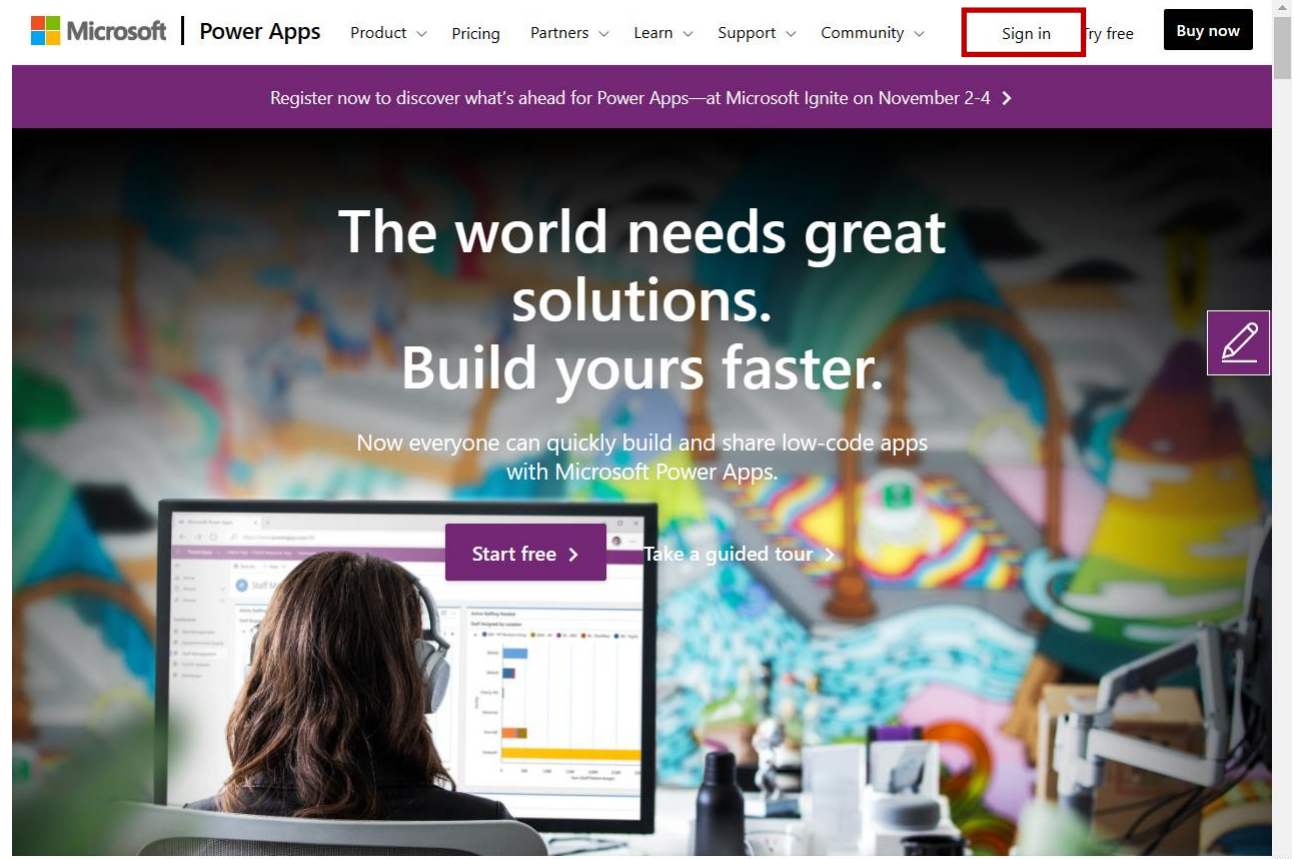


# Creating a Power Platform custom connector to wrap the Azure Function



# Step 13. Log in to Power Apps.

Got to  
<https://powerapps.microsoft.com/> and  
sign in with your  
organization account



# Step 14. Create a solution

Once signed in click **Solutions**.

Then click **+ New solution**

In the New solution pane click **+ New publisher**

Note, a publisher enables us to have multiple components with same name as their schema name will always be prefixed with the publisher prefix.

It also helps us identify components created by different publishers

When creating a solution you must select a publisher.

The screenshot shows the Power Apps interface. On the left, the 'Solutions' menu item is highlighted with a red box. In the top navigation bar, the '+ New solution' button is highlighted with a red box. On the right, the 'New solution' pane is open, showing fields for 'Display name', 'Name', 'Publisher', and 'Version'. The '+ New publisher' button in the 'Publisher' section is highlighted with a red box. The 'Create' and 'Cancel' buttons are at the bottom right.

Power Apps

Search

+ New solution Import ... Solution

Solutions

Home Learn Apps Create Dataverse Flows Chatbots AI Builder Solutions

No match found

Try searching a different term. If you still can't find it, try switching to a different environment.

New solution

Display name \*

Name \*

Publisher \*

Select a Publisher

+ New publisher

Version \*

1.0.0.0

More options ▾

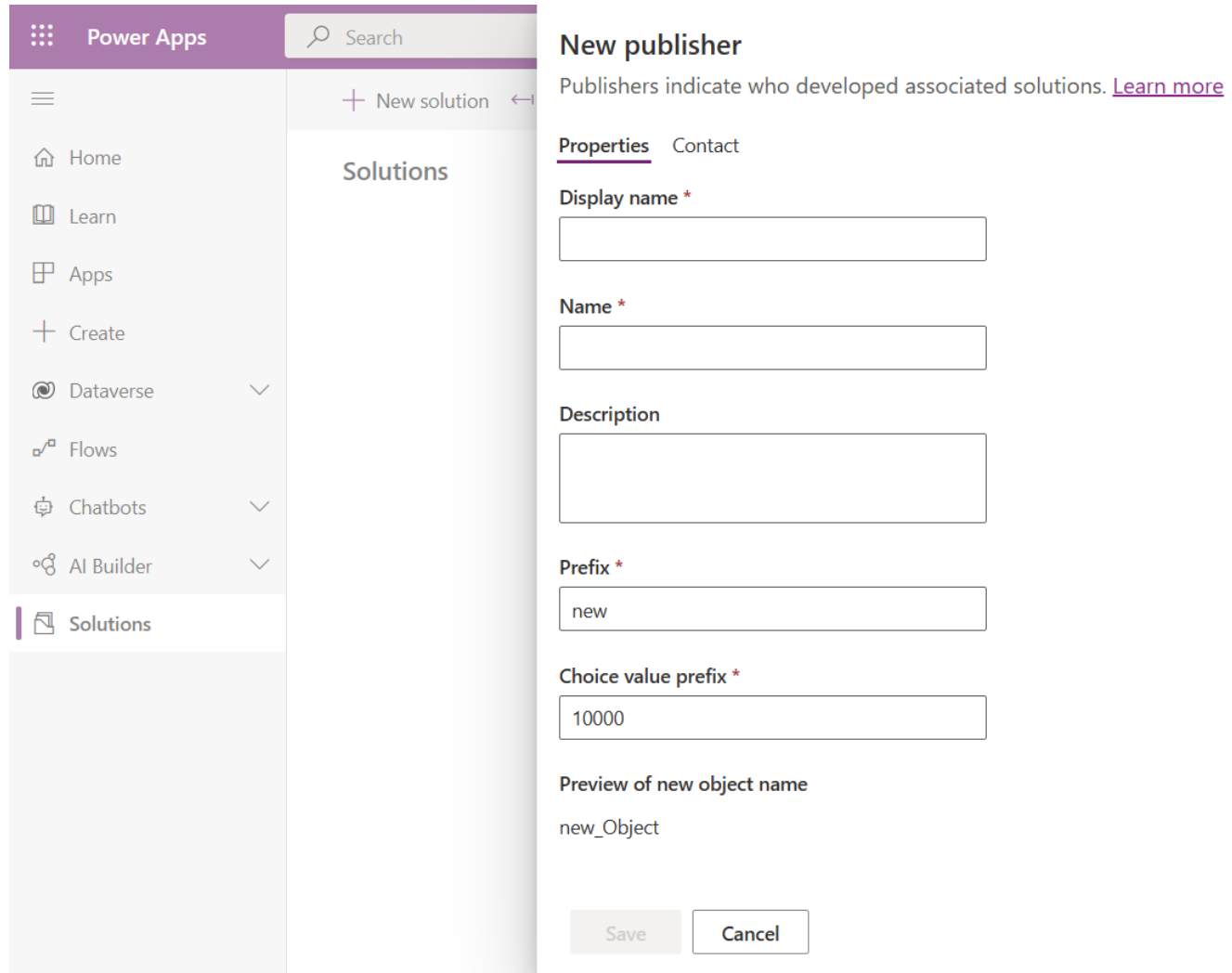
Create Cancel

# Step 15. Create a Publisher

Enter the following details:

- Display name : your name
- Name : your name with no white spaces
- Prefix: your alias
- Choise value prefix: leave as is

Click Save



The screenshot shows the 'New publisher' form in the Power Apps interface. The left sidebar contains navigation options: Home, Learn, Apps, Create, Dataverse, Flows, Chatbots, AI Builder, and Solutions (which is highlighted). The main area is titled 'Solutions' and includes a '+ New solution' button. The 'New publisher' form is open, showing the 'Properties' tab. It contains the following fields: 'Display name \*' (empty), 'Name \*' (empty), 'Description' (empty), 'Prefix \*' (containing 'new'), and 'Choice value prefix \*' (containing '10000'). Below these fields is a 'Preview of new object name' section showing 'new\_Object'. At the bottom right are 'Save' and 'Cancel' buttons. A close button (X) is in the top right corner of the form.

Power Apps Search

+ New solution

Solutions

New publisher

Publishers indicate who developed associated solutions. [Learn more](#)

Properties Contact

Display name \*

Name \*

Description

Prefix \*

Choice value prefix \*

Preview of new object name

new\_Object

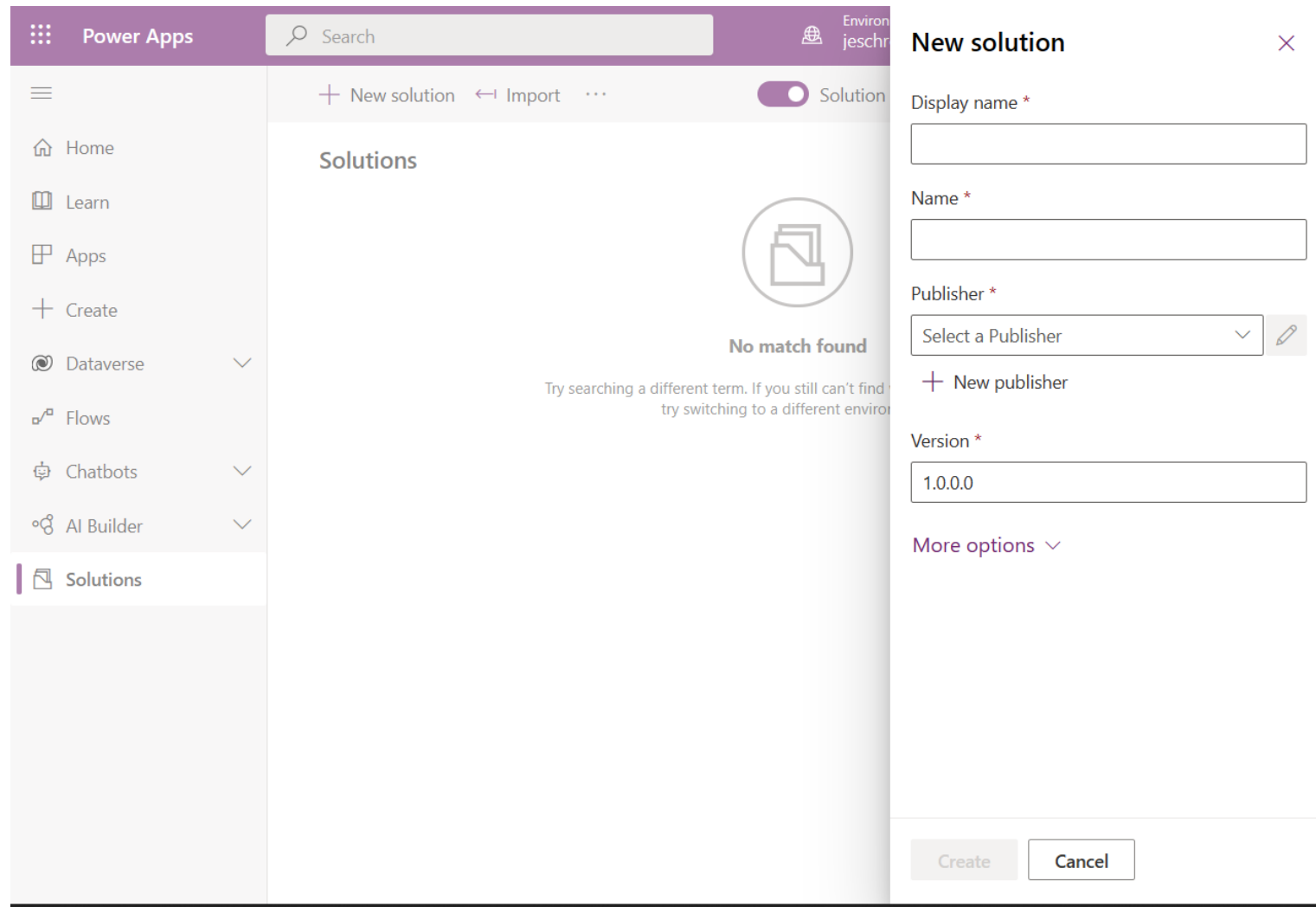
Save Cancel

# Step 16. Complete the solution creation

Back at the create solution pane, enter the following details:

- Display name: name you solution. Include your alias
- Name: leave as it is generated
- Publisher: select the publisher you just created

Click Create and click you newly created solution



The screenshot shows the Power Apps interface. On the left is a navigation pane with options: Home, Learn, Apps, Create, Dataverse, Flows, Chatbots, AI Builder, and Solutions. The main area displays 'Solutions' with a search bar and a 'No match found' message. On the right, the 'New solution' pane is open, showing fields for 'Display name', 'Name', 'Publisher' (with a dropdown menu), and 'Version' (set to 1.0.0.0). There are 'Create' and 'Cancel' buttons at the bottom right.

**Power Apps** Search

+ New solution ← Import ... Solution

**Solutions**

No match found

Try searching a different term. If you still can't find it, try switching to a different environment.

**New solution** ✕

Display name \*

Name \*

Publisher \*  
Select a Publisher

+ New publisher

Version \*  
1.0.0.0

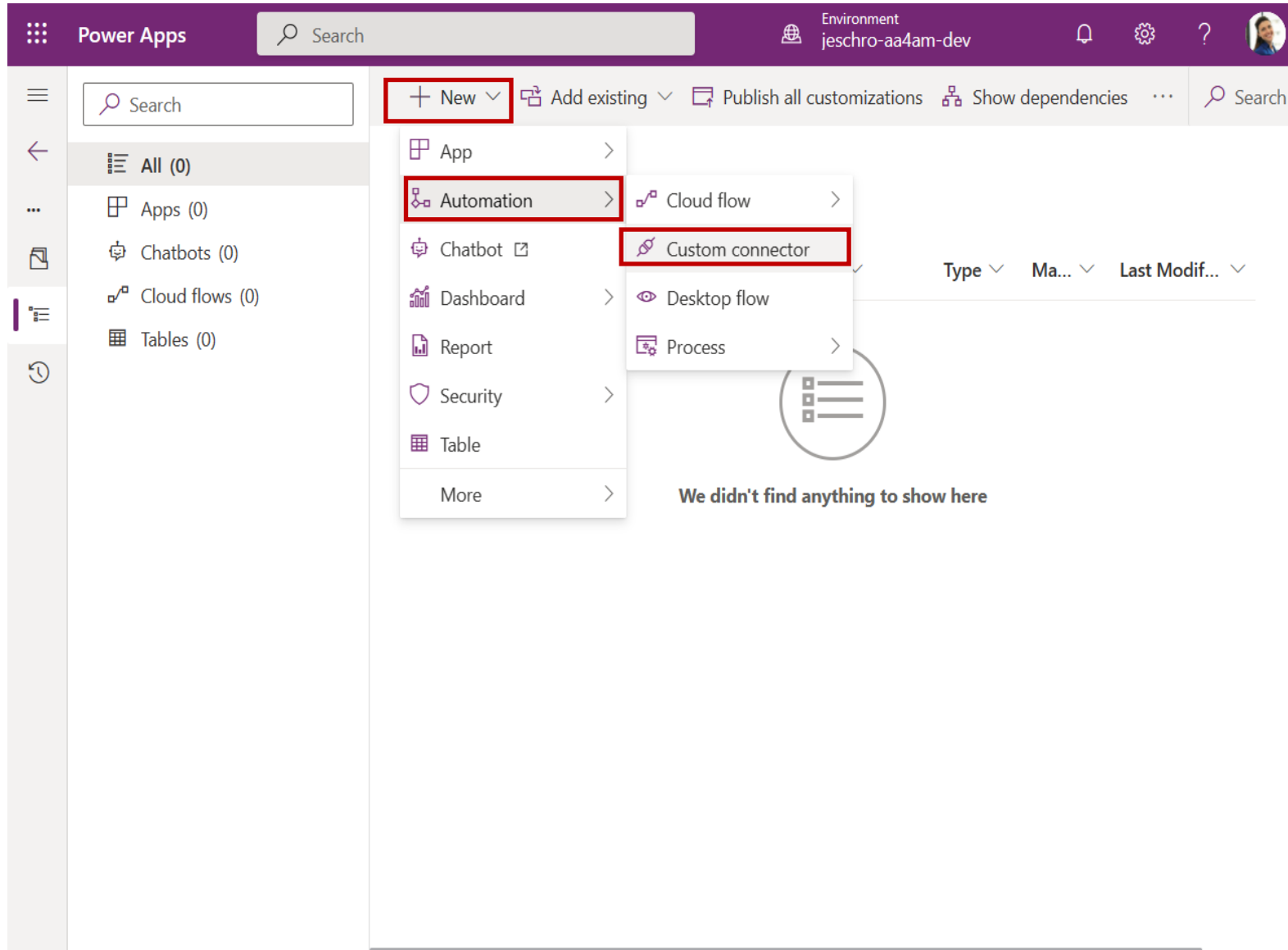
More options ▾

Create Cancel

# Step 17. Add a custom connector to your solution

Open the solution you just created.

Click New =>  
Automation =>  
Custom connector



# Step 18. Set the General properties of your custom connector

Enter the following details:

- Connector Name: alias + WeatherConnector
- Host: [your function app name].azurewebsites.net
- Base URL: /api

The screenshot shows the 'Power Automate' interface for creating a custom connector. The 'Connector Name' field is highlighted with a red box and contains the text 'Untitled'. The 'Host' field is also highlighted with a red box and contains the text 'api.contoso.com'. The 'Base URL' field is highlighted with a red box and contains the text '/'. The 'Scheme' is set to 'HTTPS'. The 'Description' field contains the text 'Give your custom connector a short description'. The 'Icon background color' field contains the text 'A color to show behind the icon (e.g., '#007ee5')'. The 'Upload connector icon' button is visible. The 'Connect via on-premises data gateway' checkbox is unchecked. The 'Swagger Editor' toggle is turned off. The 'Create connector' button is visible. The 'Cancel' button is visible. The 'Security' button is visible at the bottom right.

Power Automate

Search for helpful resources

Environments  
jeschro-aa4am-dev

Connector Name: **Untitled**


1. General > 2. Security > 3. Definition > 4. Code (Preview) > 5. Test

Swagger Editor Create connector Cancel

**General information**

Add an icon and short description to your custom connector. Your host and base URL will be automatically generated from the swagger file.

**General information**

 [Upload connector icon](#)  
Supported file formats are PNG and JPG. (< 1MB)

[Upload](#)

Icon background color

A color to show behind the icon (e.g., '#007ee5')

Description

Give your custom connector a short description

☐ Connect via on-premises data gateway [Learn more](#)

Scheme \*

☒ HTTPS ☐ HTTP

Host \*

**api.contoso.com**

Base URL

**/**

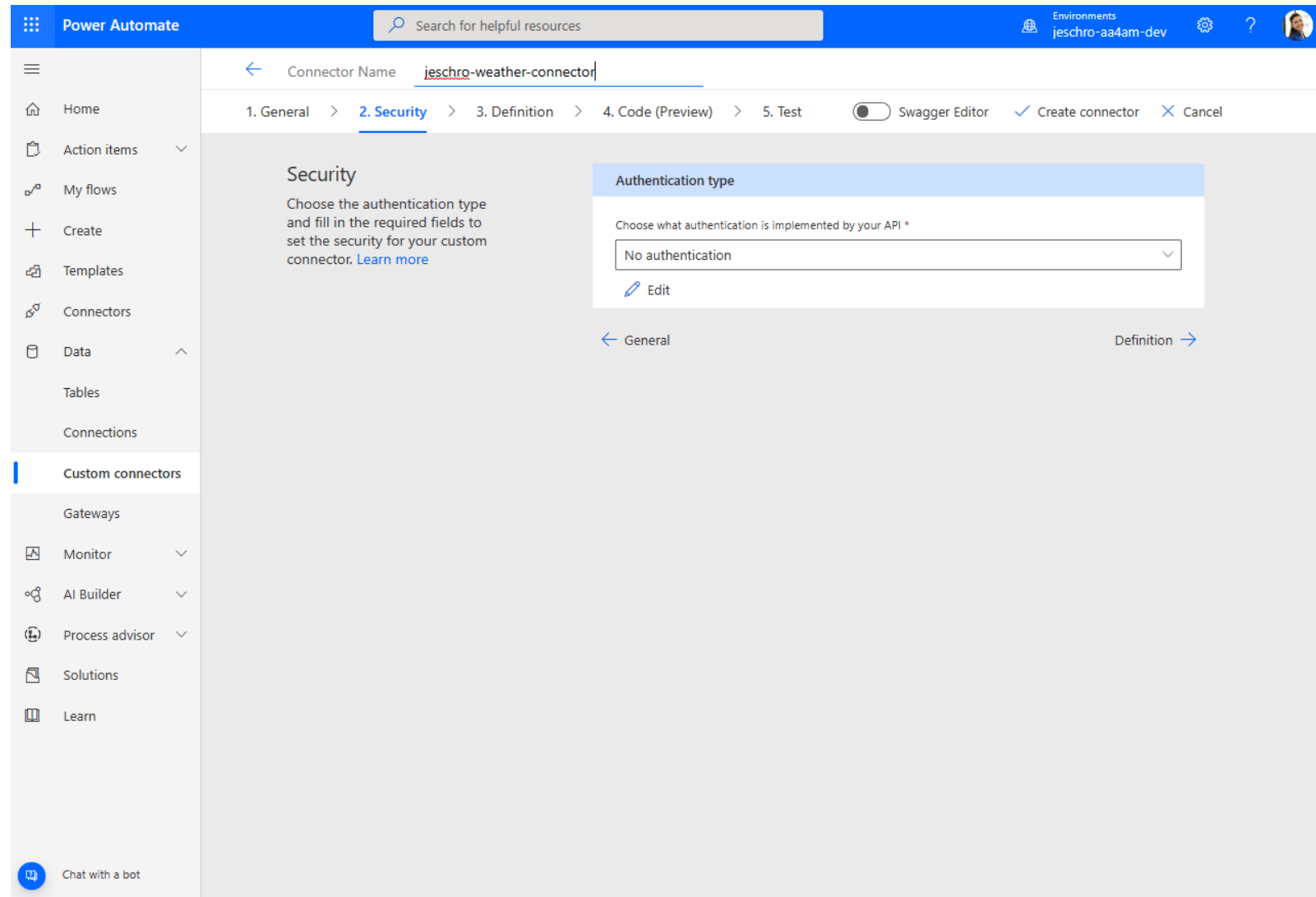
Security →

# Step 19. Security properties

In this lab we will not configure any security settings.

However, have a look at the different authentication methods that are supported.

For Azure services it is common to use OAuth 2 with Azure AD Identity Provider.

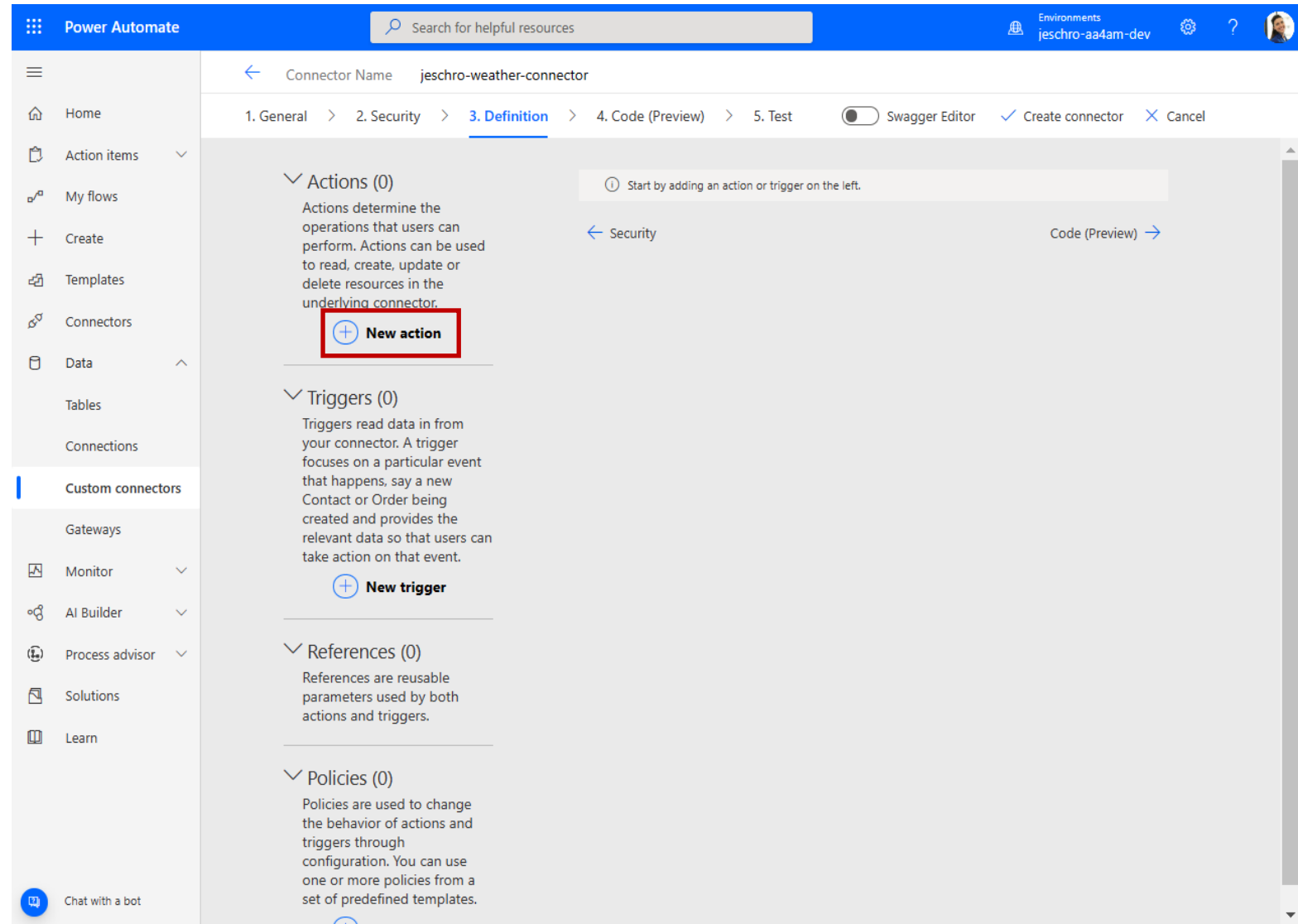


# Step 20. Definition properties

This is where we define the functionality of our connector

We must create an action to retrieve the weather forecast

Start by clicking **+New action**





# Step 21. Define the General properties of your new action

Enter the following details in the General section:

- Summary: Get weather by location
- Operation ID: GetWeatherByLocation

Power Automate

Search for helpful resources

Environments  
jeschro-aa4am-dev

Connector Name jeschro-weather-connector

1. General > 2. Security > **3. Definition** > 4. Code (Preview) > 5. Test

Swagger Editor Create connector Cancel

**Actions (1)**  
Actions determine the operations that users can perform. Actions can be used to read, create, update or delete resources in the underlying connector.  
New action

**Triggers (0)**  
Triggers read data in from your connector. A trigger focuses on a particular event that happens, say a new Contact or Order being created and provides the relevant data so that users can take action on that event.  
New trigger

**References (0)**  
References are reusable parameters used by both actions and triggers.

**Policies (0)**  
Policies are used to change the behavior of actions and triggers through configuration. You can use one or more policies from a

**General**

Summary [Learn more](#)

Description [Learn more](#)

Operation ID \*  
This is the unique string used to identify the operation.

Visibility [Learn more](#)

☒ none ☐ advanced ☐ internal ☐ important

**Request**  
It defines the pre-requirements needed in order to make a request. Describes a single operation parameter. A unique parameter is defined by a combination of a name and location.  
+ Import from sample

**Response**  
It defines the shape of response returned by the underlying connector when making the request.

default default

# Step 22. Now define the Request schema of the action

Click **+import from sample** in the Request section and fill in the following details:

- Verb: Get
- Url: `GetWeather?name={Location}`

Click **Import**

Power Automate

Search for helpful resources

Environments  
jeschro-aa4am-dev

Connector Name: jeschro-weather-connector

1. General > 2. Security > 3. Definition > 4. Code (Preview) > 5. Test

Actions (1)  
Actions determine the operations that users can perform. Actions can be used to read, create, update or delete resources in the underlying connector.

GetWeather...  
New action

Triggers (0)  
Triggers read data in from your connector. A trigger focuses on a particular event that happens, say a new Contact or Order being created and provides the relevant data so that users can take action on that event.

New trigger

References (0)  
References are reusable parameters used by both actions and triggers.

Policies (0)  
Policies are used to change the behavior of actions and triggers through configuration. You can use one or more policies from a

General

Summary [Learn more](#)  
Get weather by location

Description [Learn more](#)

Operation ID \*  
This is the unique string used to identify the connector operation.

GetWeatherByLocation

Visibility [Learn more](#)  
☒ none ☐ advanced ☐ internal

Request

It defines the pre-requirements needed in parameter. A unique parameter is defined

+ Import from sample

Response

It defines the shape of response returned to

default default

Import from sample

Verb \*  
☒ GET ☐ DELETE ☐ POST ☐ PUT ☐ HEAD ☐ OPTIONS  
☐ PATCH

URL \*  
E.g. https://flow.microsoft.com/templates/

Headers

Headers separated by a new line, e.g.:  
Content-Type application/json  
Accept application/json

Body

JSON object with body, e.g.:  
{  
 "email": "test@test.com",  
 "name": "Jane Doe"  
}

The body is the payload that's appended to the HTTP request. There can only be one body parameter.

Import Close

# Step 23. Now define the Request schema of the action - Continued

Since we have not implemented code to handle empty/null values in our name parameter we should set the name property to required

Click the **name => Edit** in the Query section of the Request section.

Set **Is required?** to **Yes**

Click **<- Back** just above the Parameter section

The screenshot shows the Power Automate interface for defining a custom connector. The connector name is 'jeschro-weather-connector'. The 'Definition' tab is active, showing the 'Parameter' section. The 'Name' field is 'name'. The 'Is required?' checkbox is checked. The 'Location' is set to 'Query'. The 'Type' is set to 'string'.

Power Automate

Search for helpful resources

Environments  
jeschro-aa4am-dev

Connector Name: jeschro-weather-connector

1. General > 2. Security > 3. Definition > 4. Code (Preview) > 5. Test

Swagger Editor Update connector Close

Actions (1)  
Actions determine the operations that users can perform. Actions can be used to read, create, update or delete resources in the underlying connector.  
GetWeather...  
New action

Triggers (0)  
Triggers read data in from your connector. A trigger focuses on a particular event that happens, say a new Contact or Order being created and provides the relevant data so that users can take action on that event.  
New trigger

References (0)  
References are reusable parameters used by both actions and triggers.

Policies (0)  
Policies are used to change the behavior of actions and triggers through configuration. You can use one or more policies from a

Parameter

Name \*

name

Description [Learn more](#)

Summary [Learn more](#)

Default value

Is required?  
☒ Yes ☐ No

Visibility [Learn more](#)  
☒ none ☐ advanced ☐ internal ☐ important

Location \*  
☐ Path ☒ Query ☐ Header ☐ Body

Type  
string

Format

# Step 24. Now define the Response object

Scroll down to the Response section and click **+ Add default response**

The screenshot shows the 'Definition' tab of a custom connector in Power Automate. The connector name is 'jeschro-weather-connector'. The left sidebar shows the navigation menu with 'Custom connectors' selected. The main area is divided into sections: Query, Headers, Body, Response, and Validation. The 'Response' section is highlighted, showing a 'default' response type. A red box highlights the '+ Add default response' button. The 'Validation' section shows a 'Description not defined' error.

Power Automate

Search for helpful resources

Environments  
jeschro-aa4am-dev

Connector Name  
jeschro-weather-connector

1. General > 2. Security > 3. Definition > 4. Code (Preview) > 5. Test

Swagger Editor Create connector Cancel

Query  
Query parameters are appended to the URL. For example, in /items?id=####, the query parameter is id.

name ...

Headers  
These are custom headers that are part of the request.

Body  
The body is the payload that's appended to the HTTP request. There can only be one body parameter.

Response  
It defines the shape of response returned by the underlying connector when making the request.

default default

+ Add default response

Validation  
This helps you identify potential issues with this action.

Validation

Description  
Description not defined.

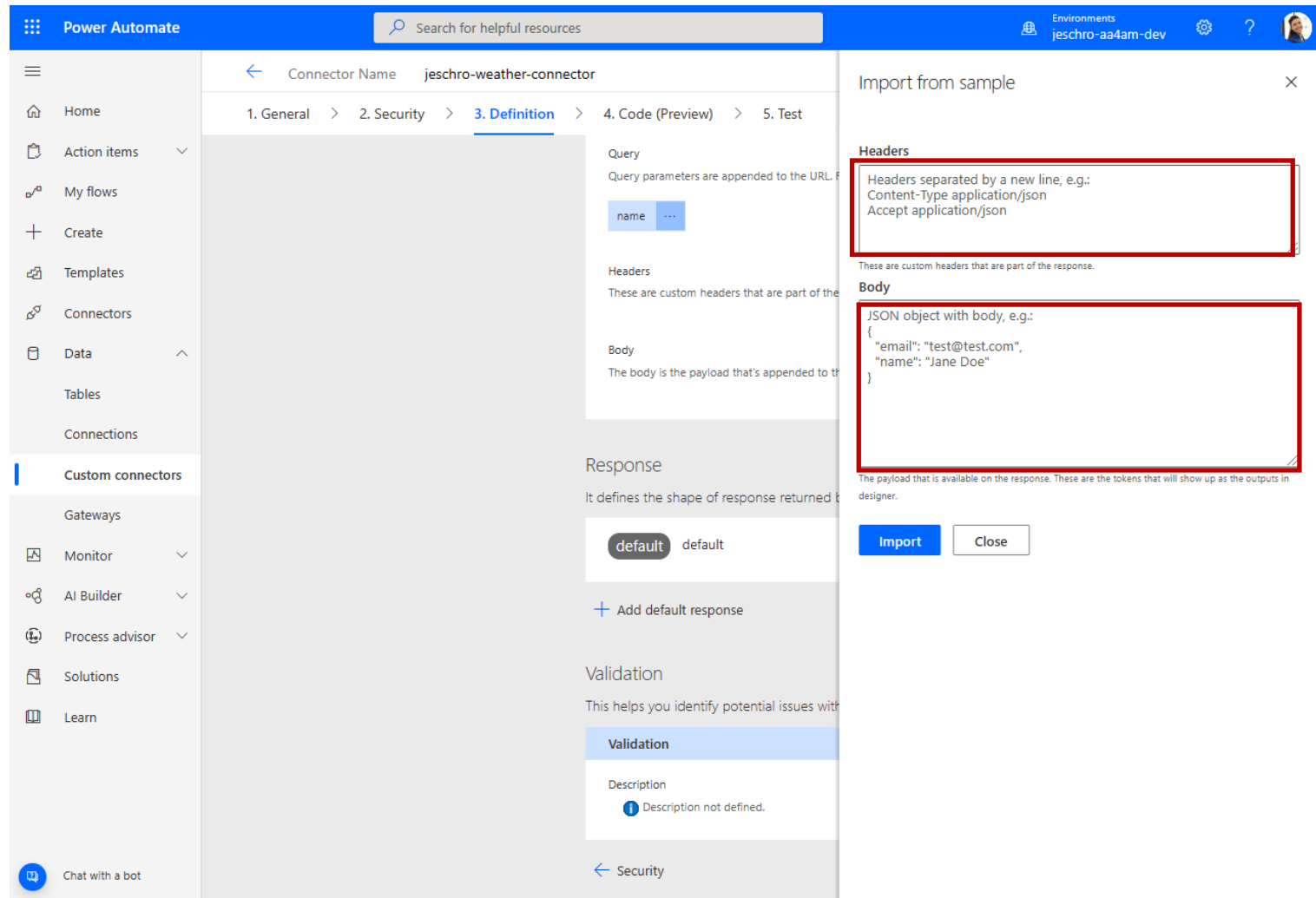
Security Code (Preview)

# Step 25. Continue defining the Response object

Enter the following details:

- Headers: Content-Type application/json
- Body: Paste in the output of your function as a sample:  
`{"name":"Oslo","forecast":"Cloudy"}`

Click **Import**



# Step 26. Continue past the Code (Preview) configuration

In this Lab we will not implement Code in the connector

Custom Connector Code is a Preview feature that allows you to implement advanced logic via C# code

[Write code in a custom connector | Microsoft Docs](#)

The screenshot displays the Power Automate web interface. The top navigation bar includes the 'Power Automate' logo, a search bar, and environment information ('jeschro-aa4am-dev'). The left sidebar shows a navigation menu with options like Home, Action items, My flows, Create, Templates, Connectors, Data, Tables, Connections, Custom connectors, Gateways, Monitor, AI Builder, Process advisor, Solutions, and Learn. The main content area is titled 'Connector Name: jeschro-weather-connector' and shows a breadcrumb trail: '1. General > 2. Security > 3. Definition > 4. Code (Preview) > 5. Test'. The 'Code (Preview)' step is active, with a toggle for 'Swagger Editor' and buttons for 'Update connector' and 'Close'. The 'Code (Preview)' section explains that this step is optional for custom code that transforms request and response payloads. It provides instructions on how to paste or upload code, noting that the code must be in C#, have a maximum execution time of 5 seconds, and cannot exceed 1MB. A 'Code Details' panel on the right shows a C# script for a 'Script' class, which overrides the 'ExecuteAsync' method to return a 200 OK response with a JSON body containing 'Hello World'. The script is as follows:

```
1 public class Script : ScriptBase
2 {
3     public override async Task<HttpStatusCode> ExecuteAsync()
4     {
5         HttpResponseMessage response = new HttpResponseMessage(HttpStatusCode.OK);
6         response.Content = CreateJsonContent("{\"message\": \"Hello World\"}");
7         return response;
8     }
9 }
10
```

At the bottom of the interface, there are navigation arrows for 'Definition' and 'Test'.

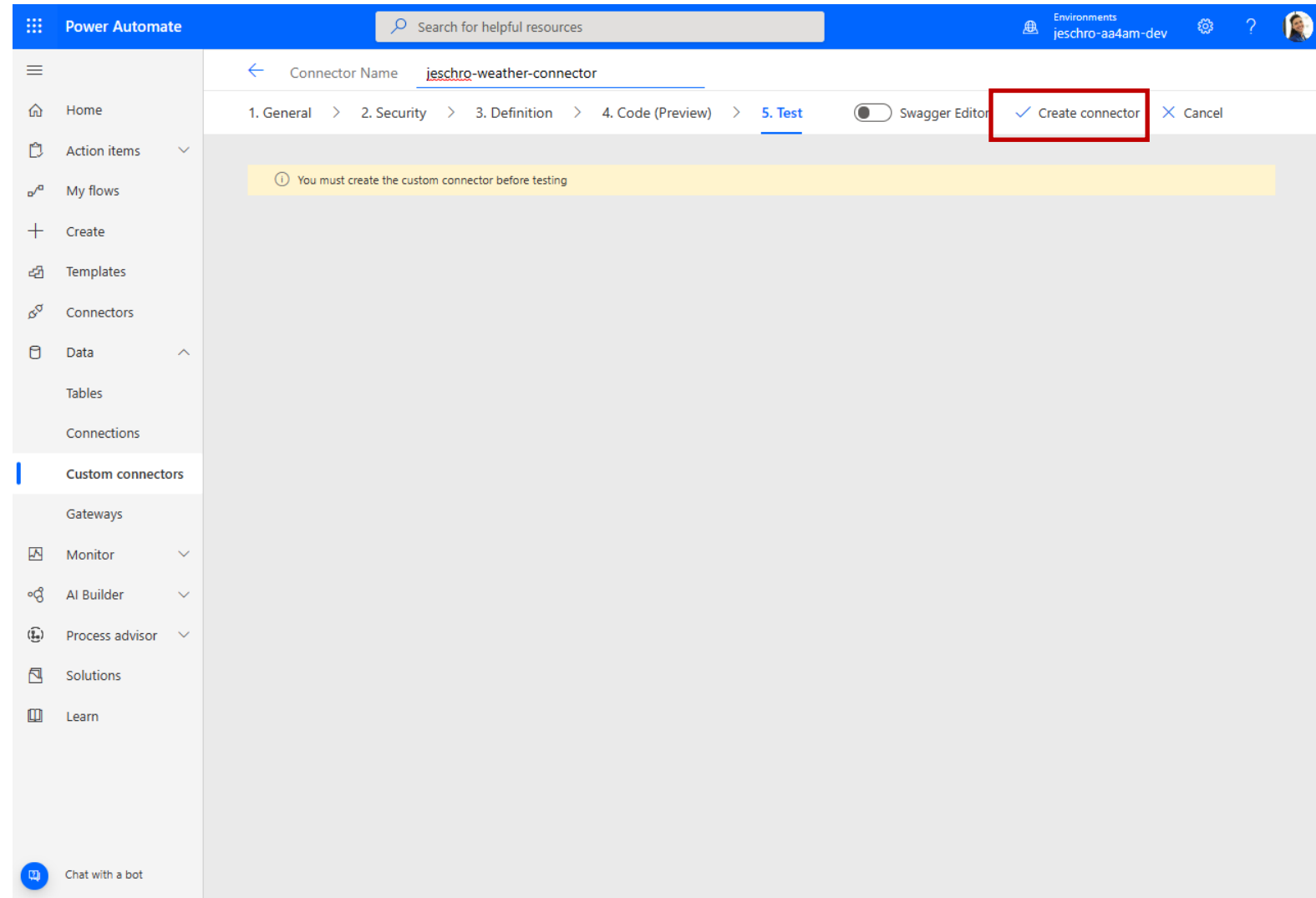
# Step 27. Create/Update your Custom Connector

Before we can test the connector it must be created first.

Click **Create connector**

If you have already created you connector make sure you update it with the latest changes you have made.

Note, it will take a short period of time for the connector to be provisioned in the supporting infrastructure



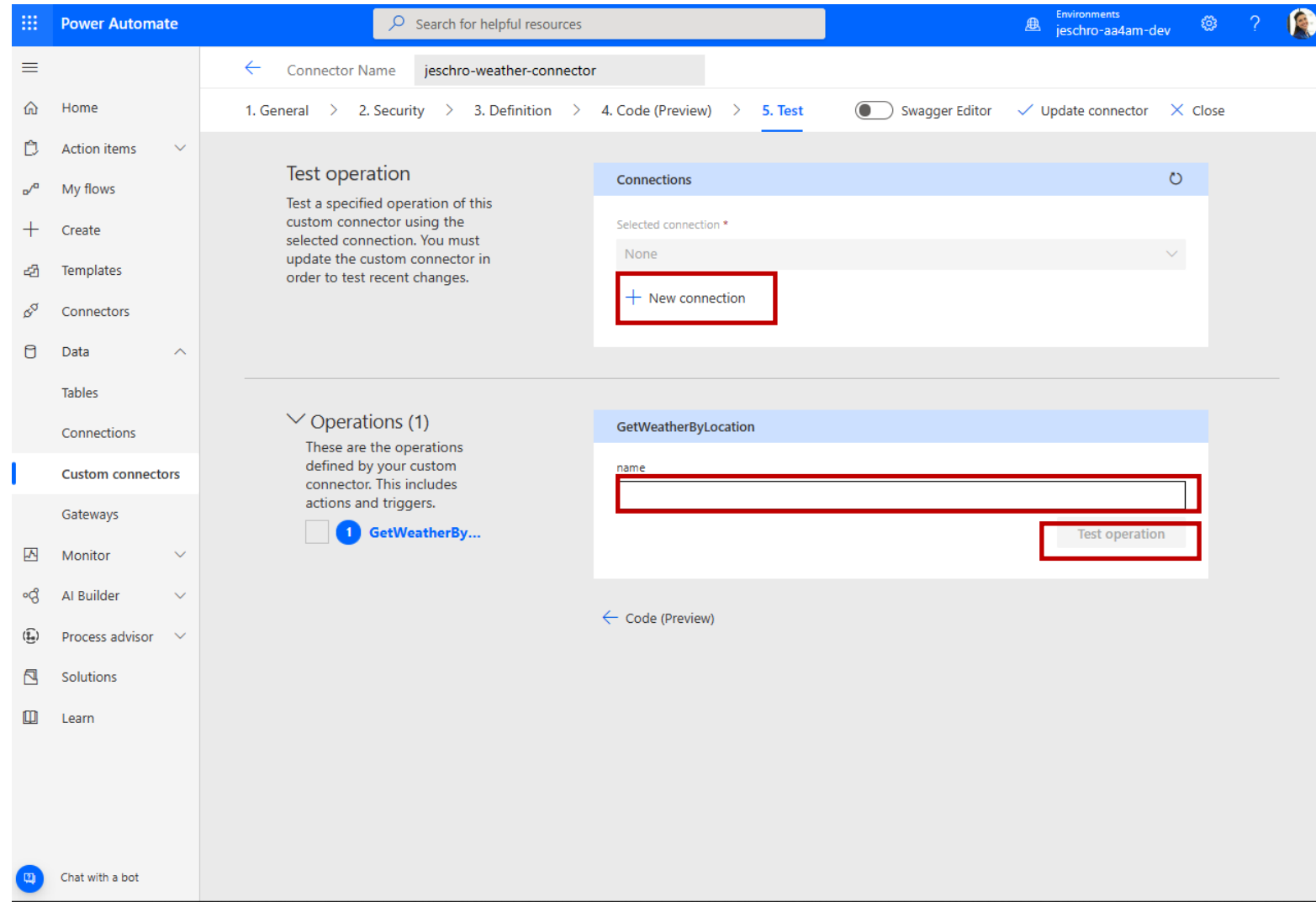
# Step 28. Test your Custom Connector

In order to use your customer connector you must first create a connection.

Click **+ New connection**

Note, because we havent implemented any authentication mechanism the connection will be created without user interaction. If authentication was implemented the user would be required to authenticate (username + password, OAuth sign in or API Key) to create a connection.

Enter a value in the name property and click **Test operation**





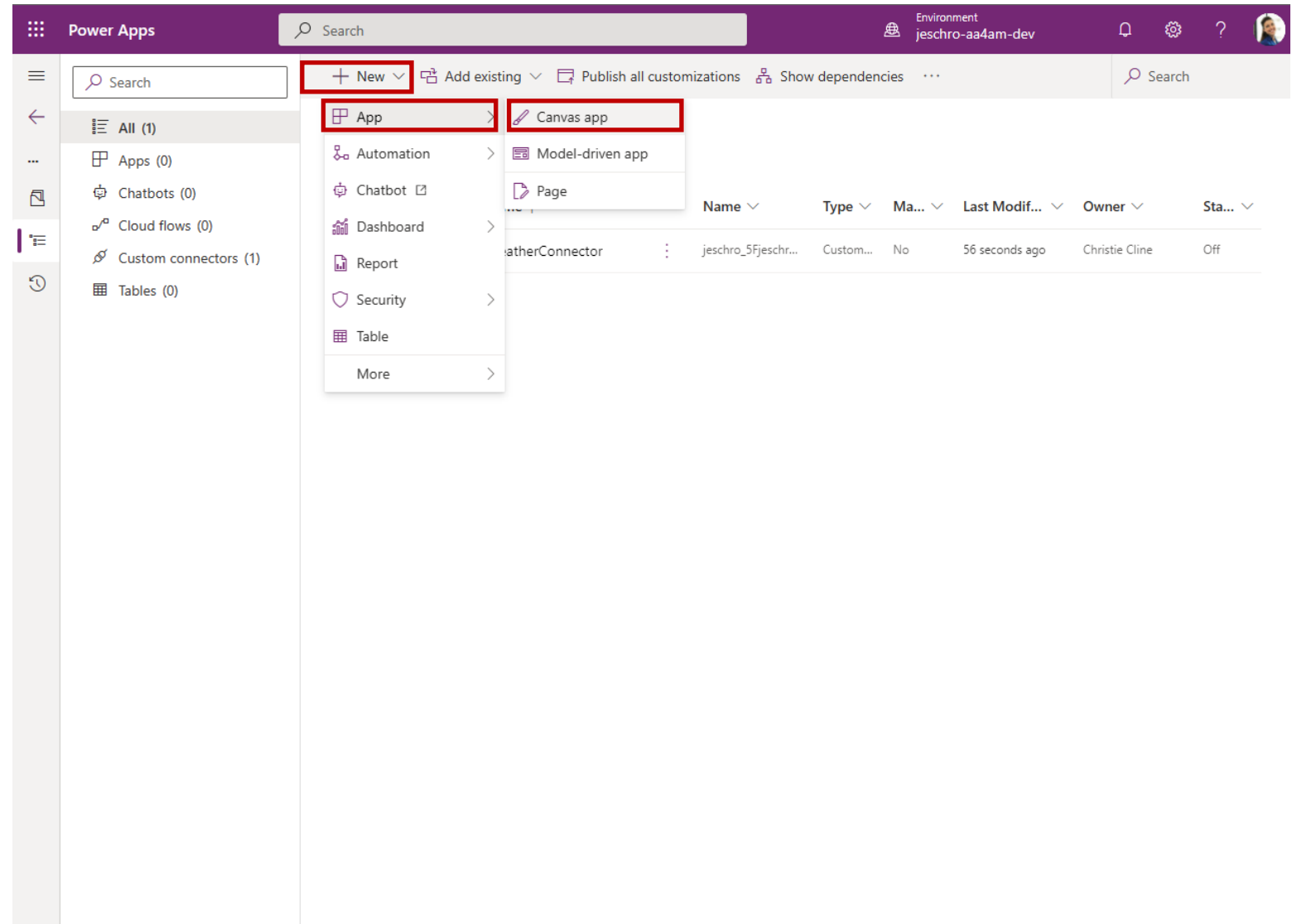
# Using a Power Platform custom connector in a Power App

# Step 29. Create a new Canvas App

Navigate to the solution you created earlier.

Notice that the custom connector has been added

Click **+New => App  
=> Canvas app**



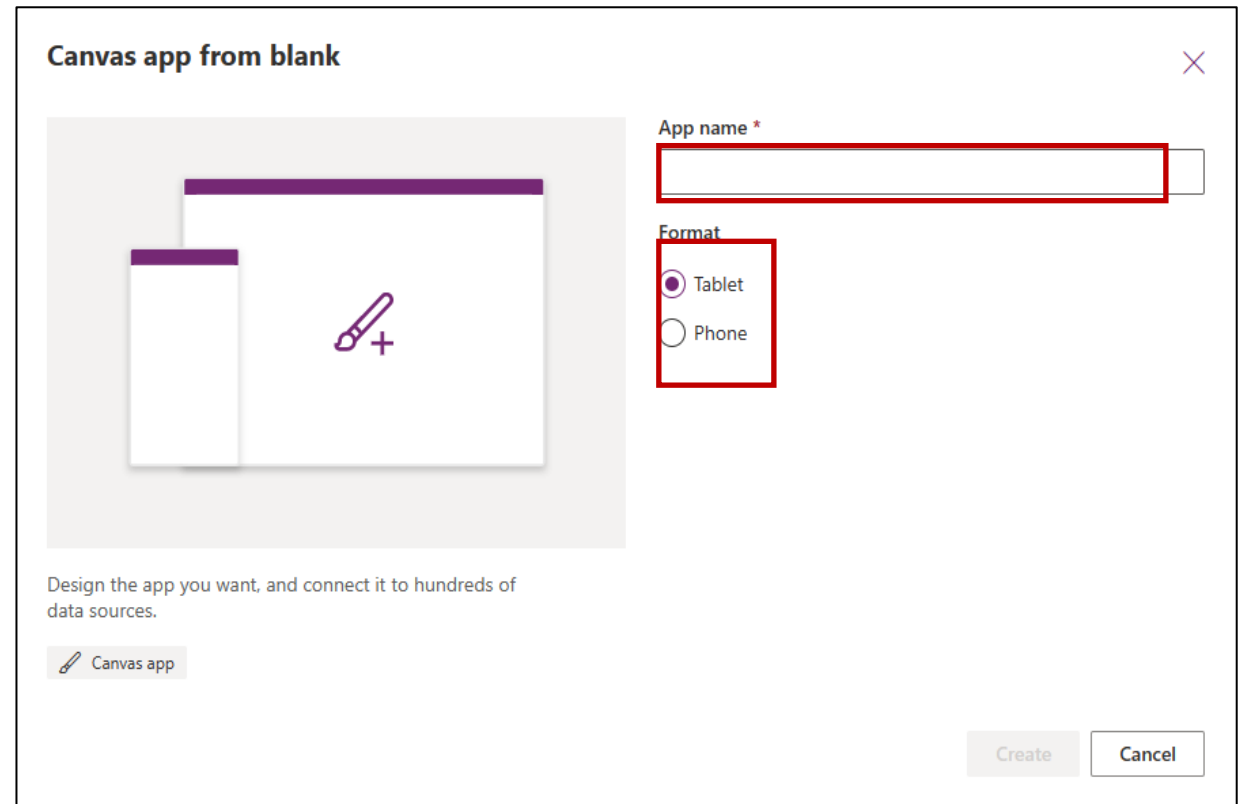
# Step 30. Name you app and select format

Give your app a name and select the format of your app

Name: Your alias + WeatherForecast

Format: Phone

Click **Create**



Canvas app from blank


App name \*

Format

☒ Tablet

☐ Phone

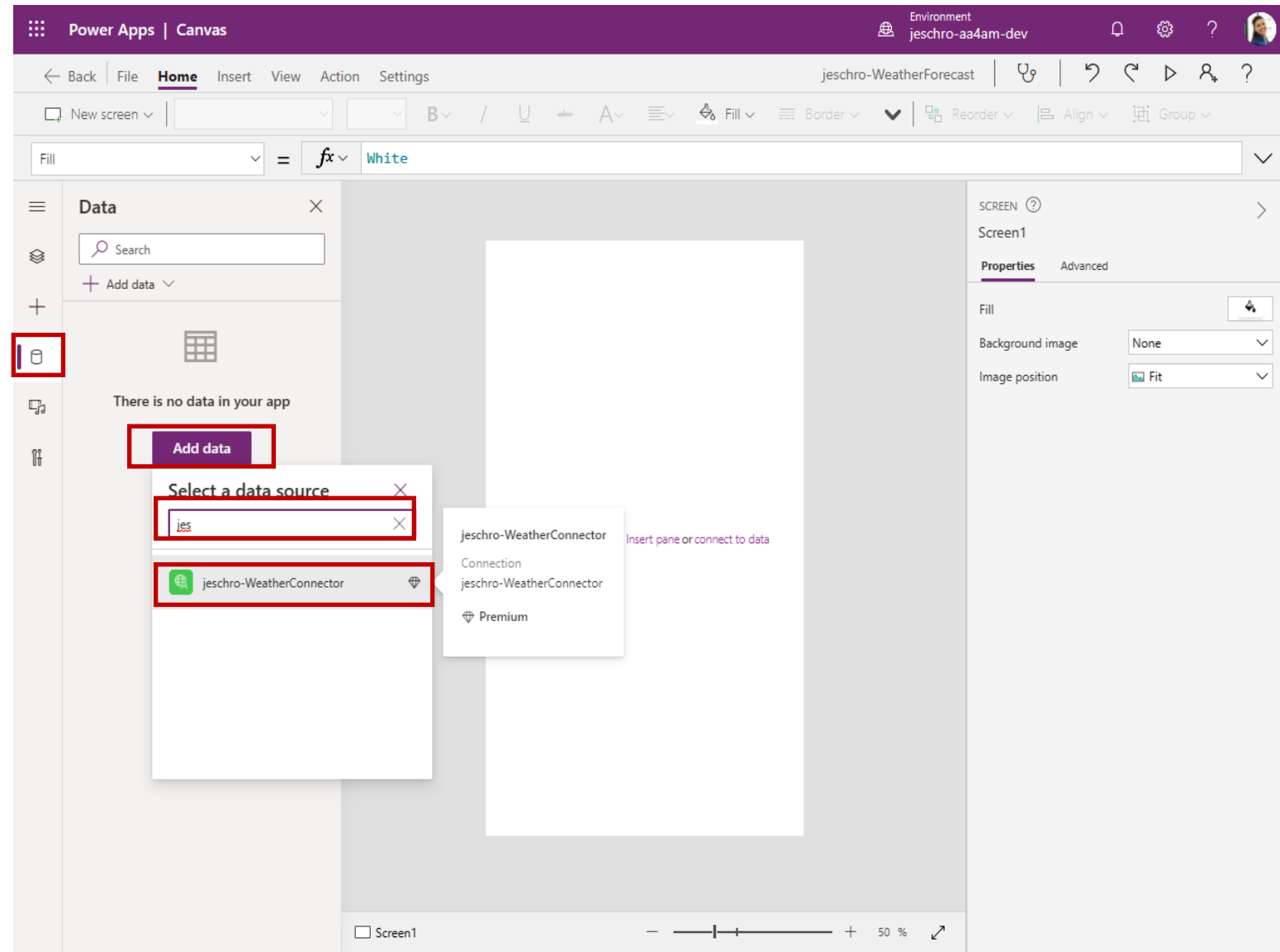
Design the app you want, and connect it to hundreds of data sources.

 Canvas app

Create Cancel

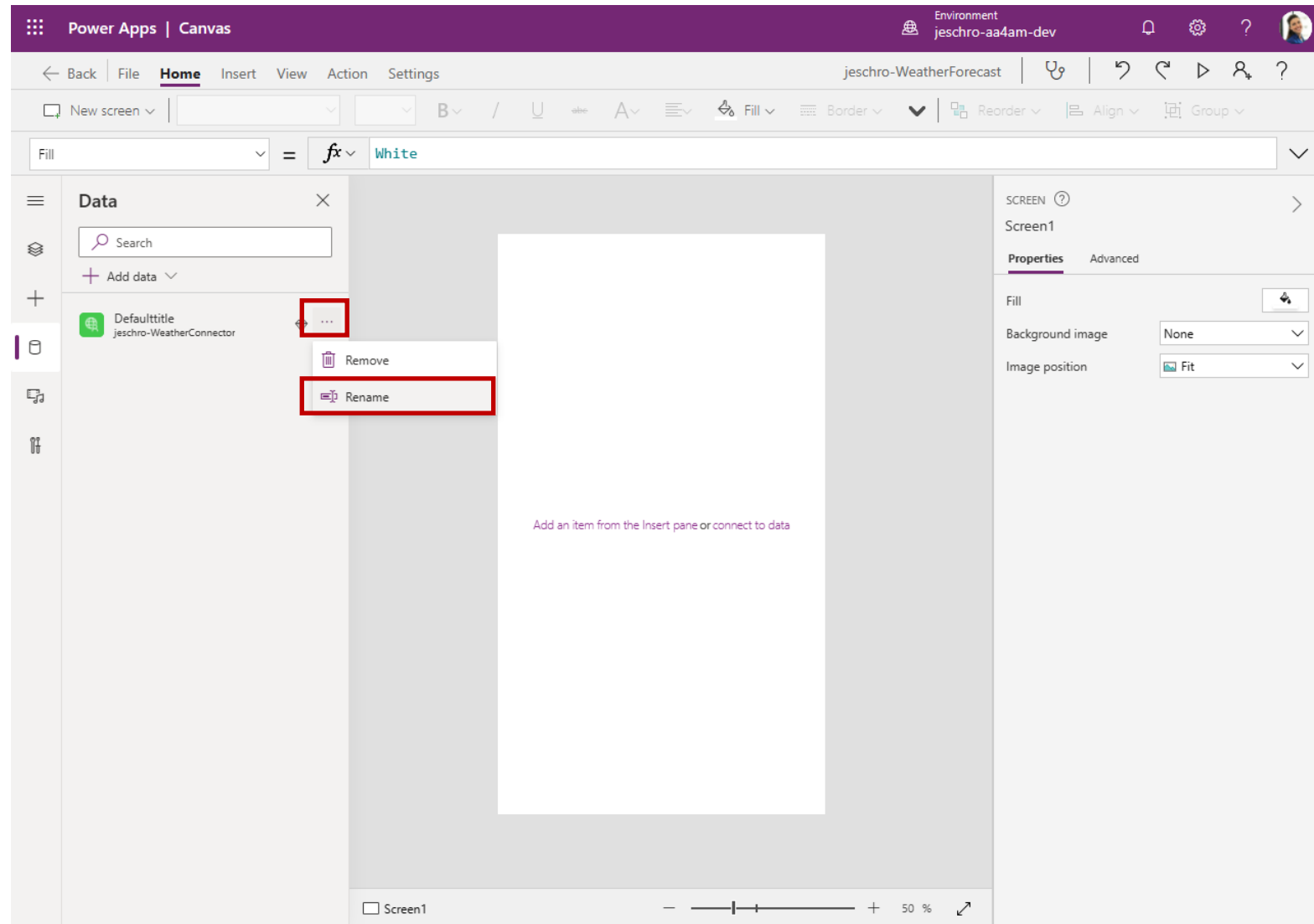
# Step 31. Add your custom connector as a datasource

1. Click the database icon on the left navigation bar
2. Click **Add data**
3. Search for your alias to find your connector
4. Click your connector and click it again



# Step 32. Rename the reference to your connector

Your connector is added with Defaulttitle as name. Change this to WeatherConnector

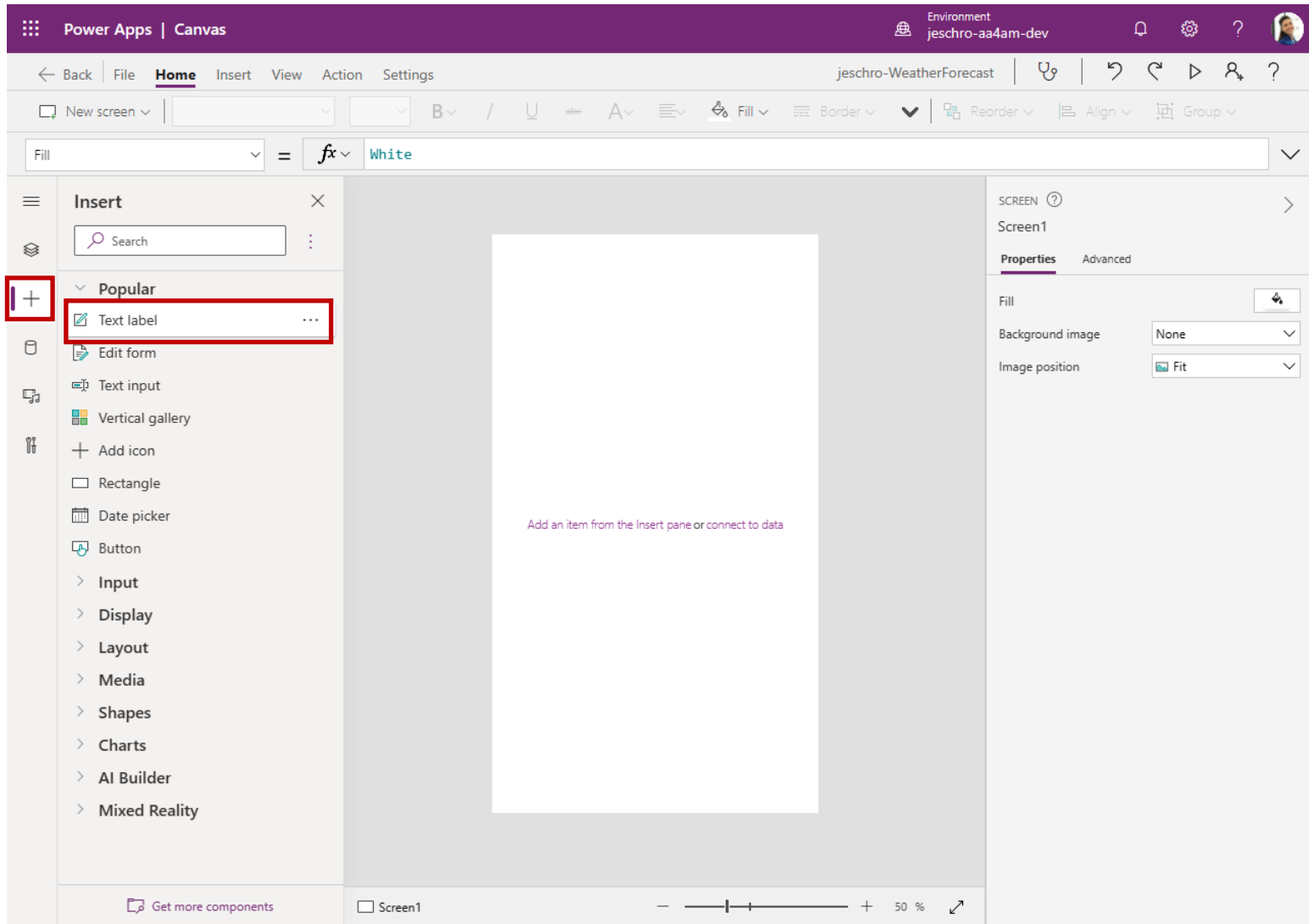


# Step 33. Add controls to your app

Add a label to your app

Click the + icon in the left Navigation bar

Click Text label



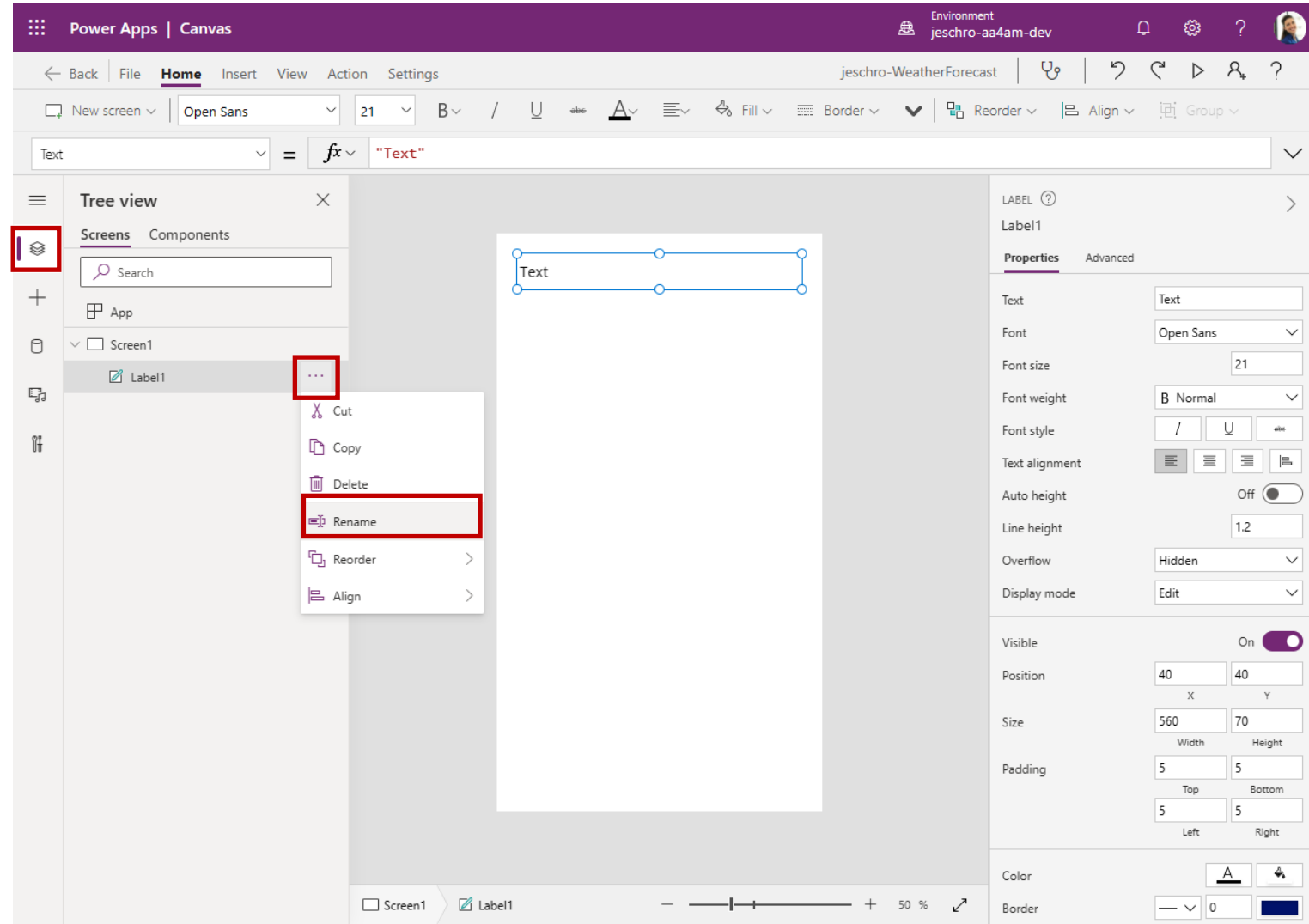
# Step 34. Set the label control properties

It is best practice to use meaningful names for your controls

Click the Treeview button (☰) in the left Navigation bar.

Click the ... next to your label control and click **Rename**

A good name could be **lblLocation**. This shows it is a label and refer to its purpose

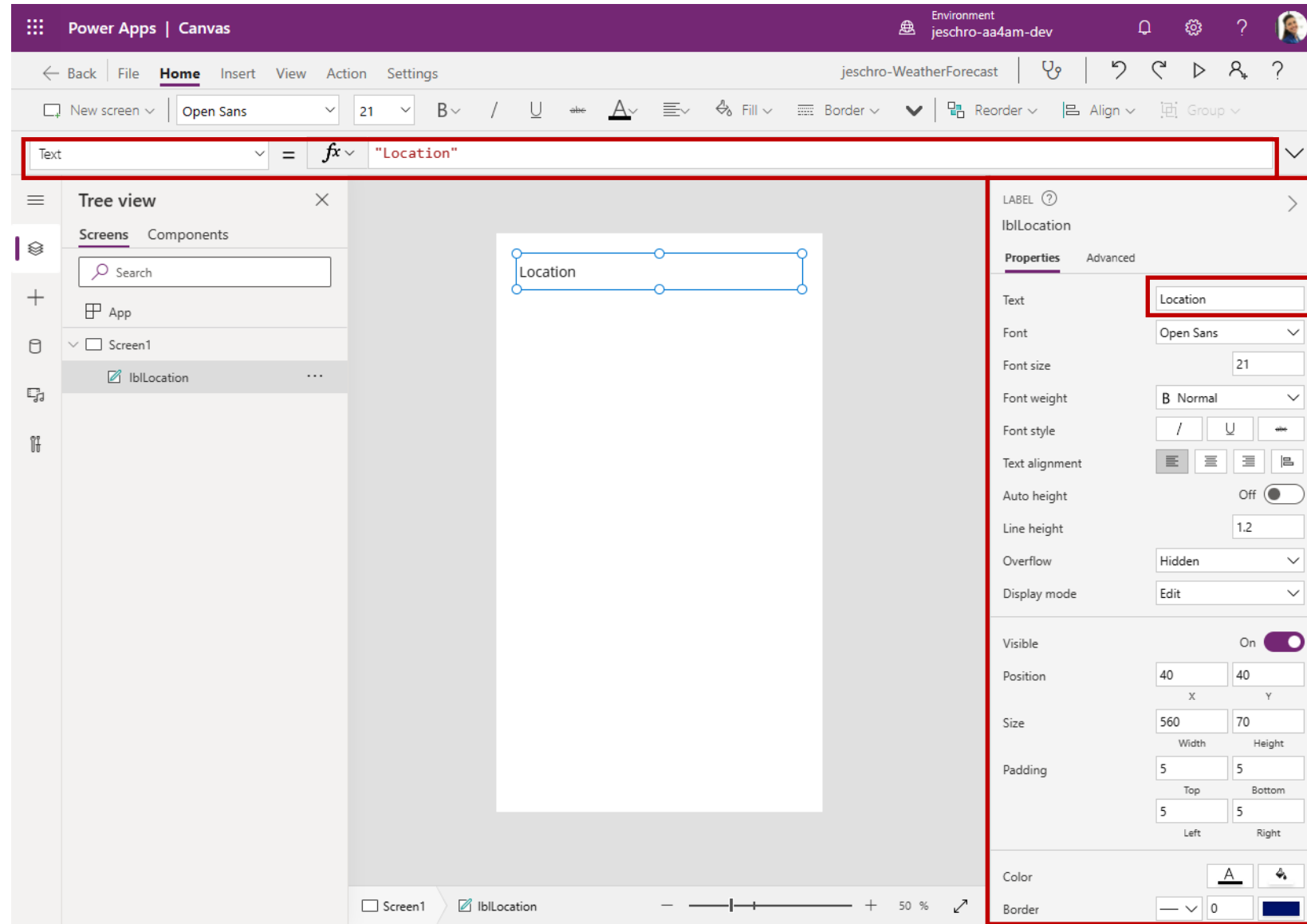


# Step 35. Set the label control properties continued

Next set the Text property of the label.

You can do this in multiple ways. Most common are via the Properties pane to the right or the Properties dropdown in top left.

Set the Text property to **Location**





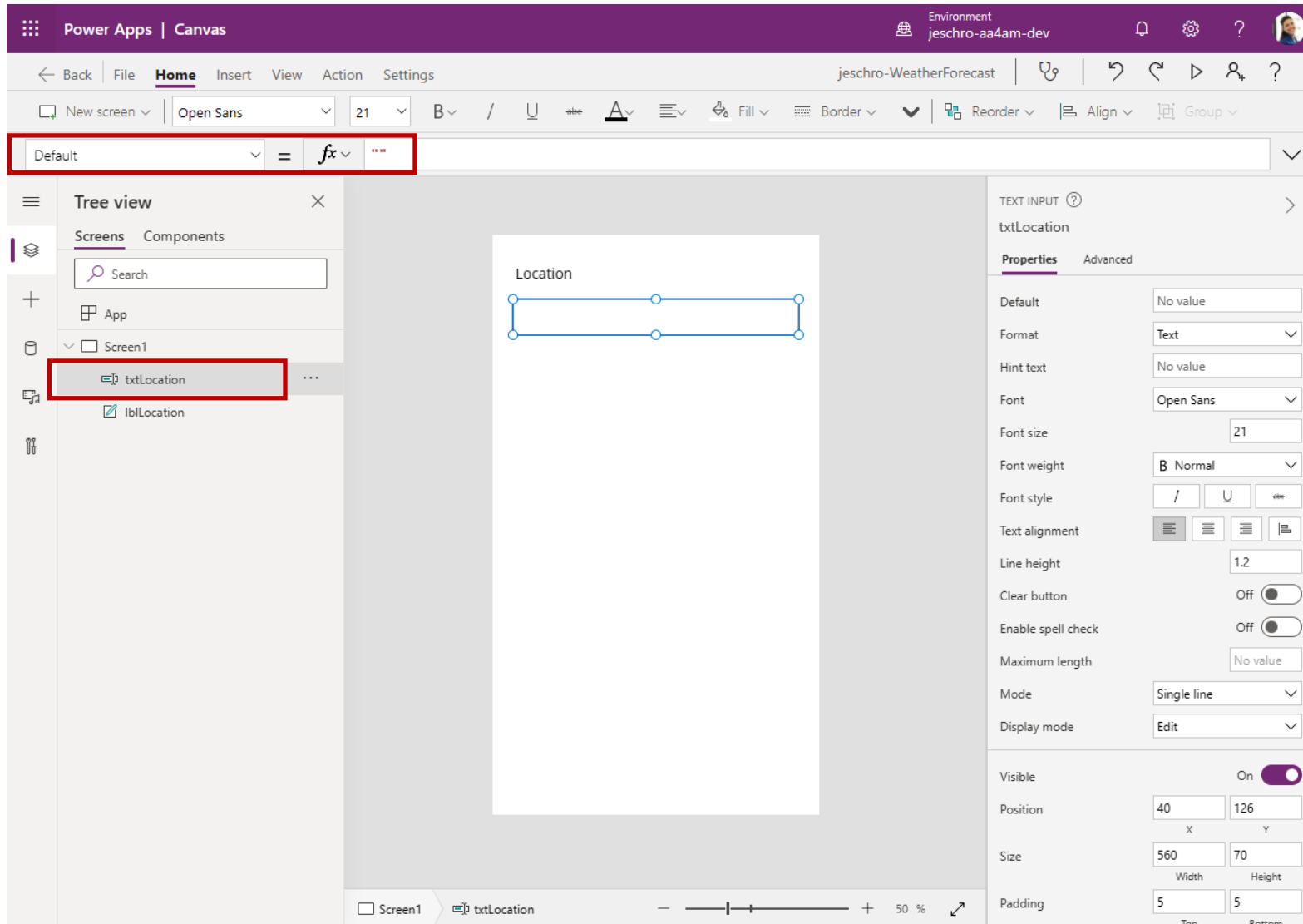
# Step 36. Add Text input control to your app

Set the following properties for the Text Input control

Name: txtLocation

Default: ""

Drag the control to just below the lblLocation label on the screen



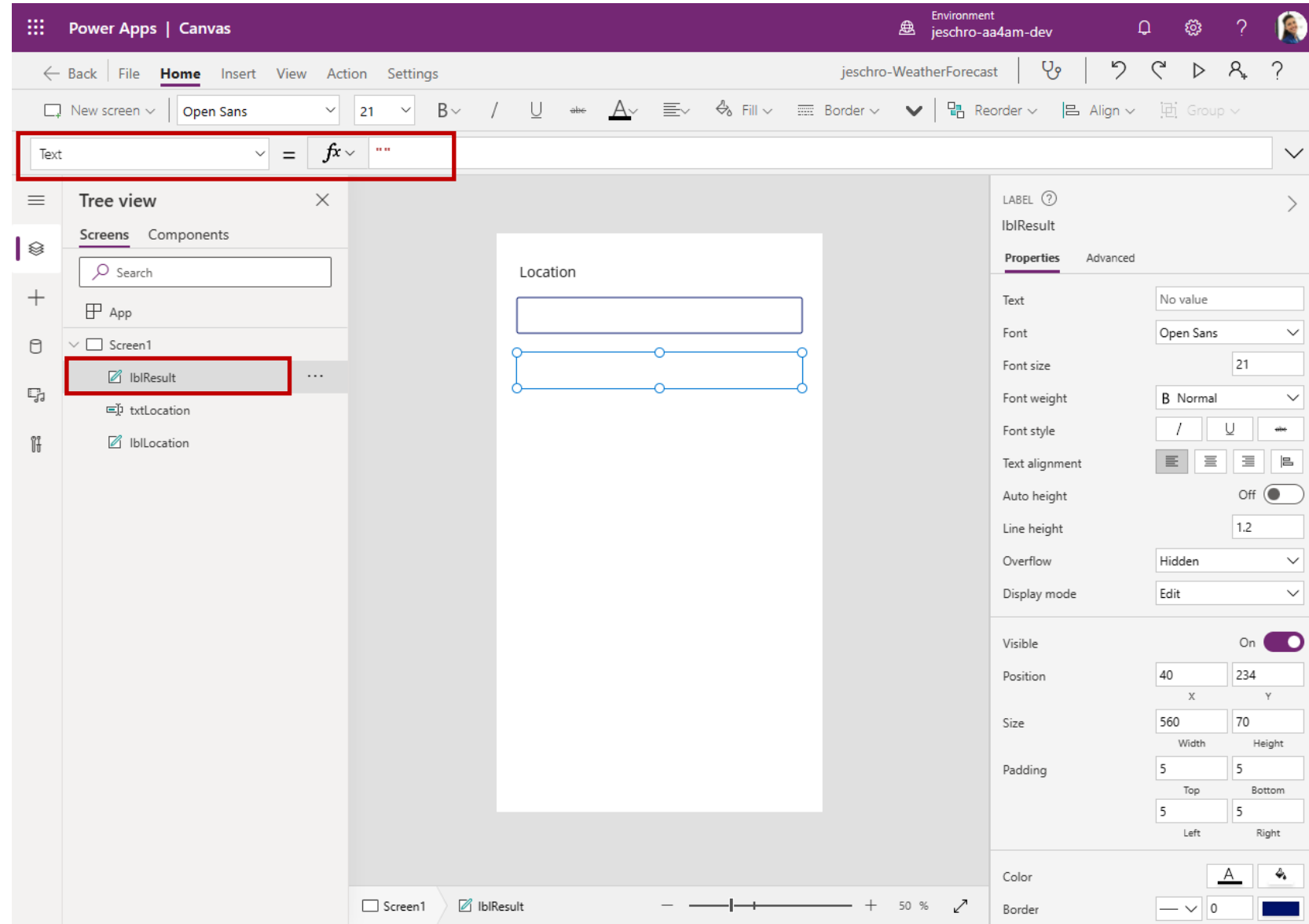
# Step 37. Add label to display the forecast result

Set the following properties for the label control

Name: lblResult

Text: ""

Drag the control to just below the txtLocation input on the screen



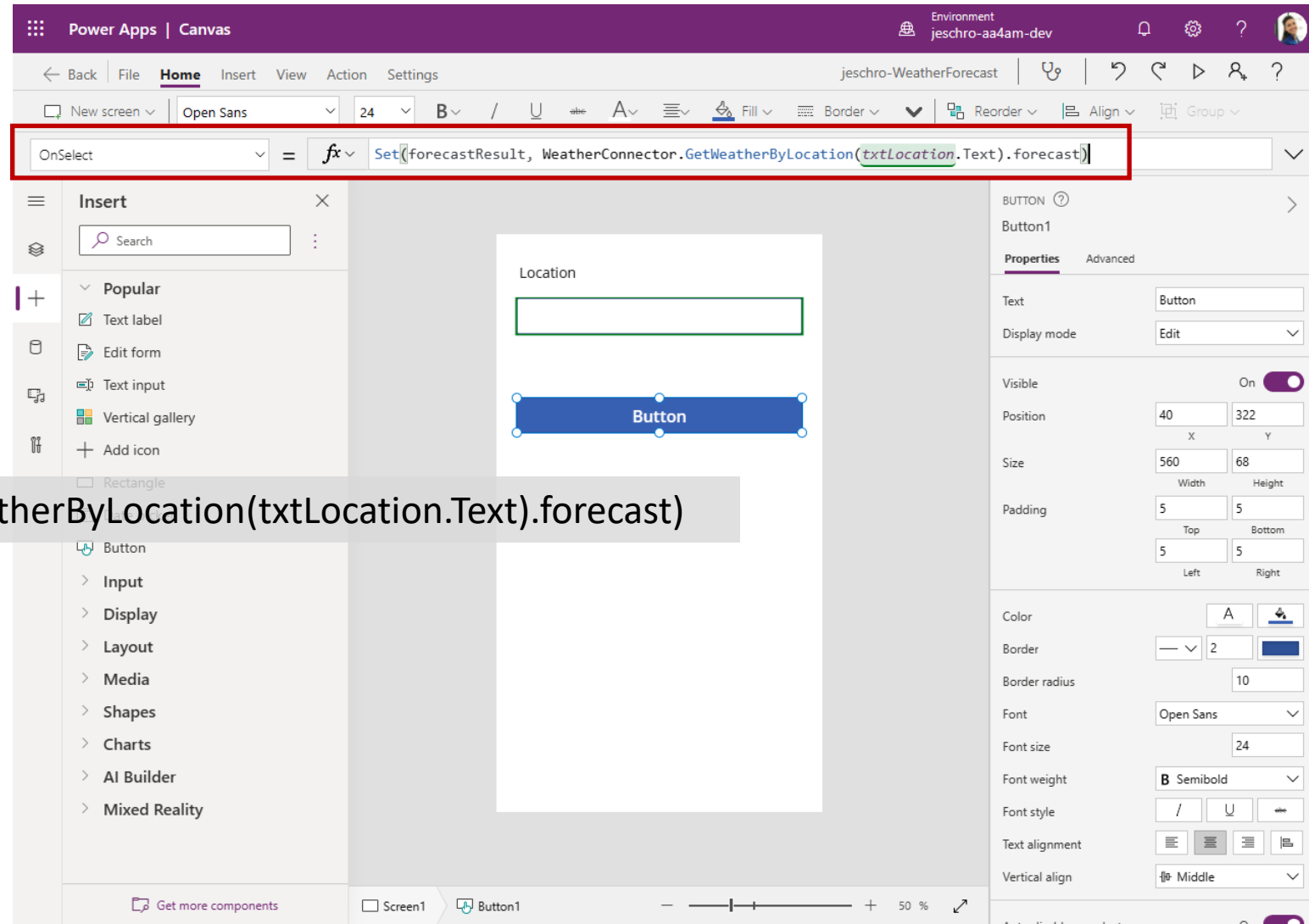
# Step 38. Add a button to call our custom connector

Add a button and drag it below the lblResult label on the screen

Select the **OnSelect** property in the property dropdown and enter the following formula:

```
Set(forecastResult, WeatherConnector.GetWeatherByLocation(txtLocation.Text).forecast)
```

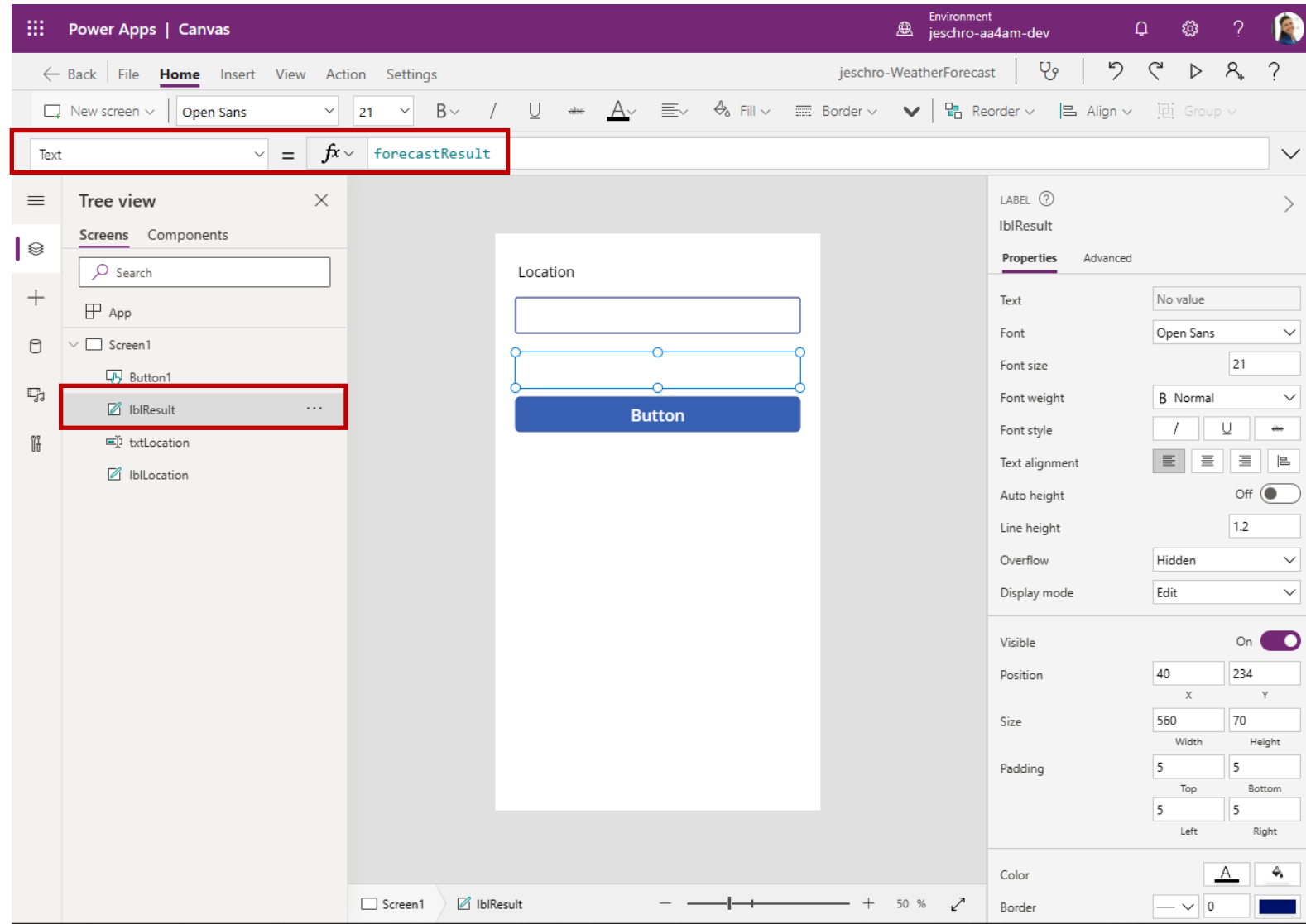
This will set a variable called forecastResult to the forecast property of the object returned by the WeatherConnector



# Step 39. Display the result in the result label

Select the lblResult label in the Tree view and set its **Text** property to **forecastResult**

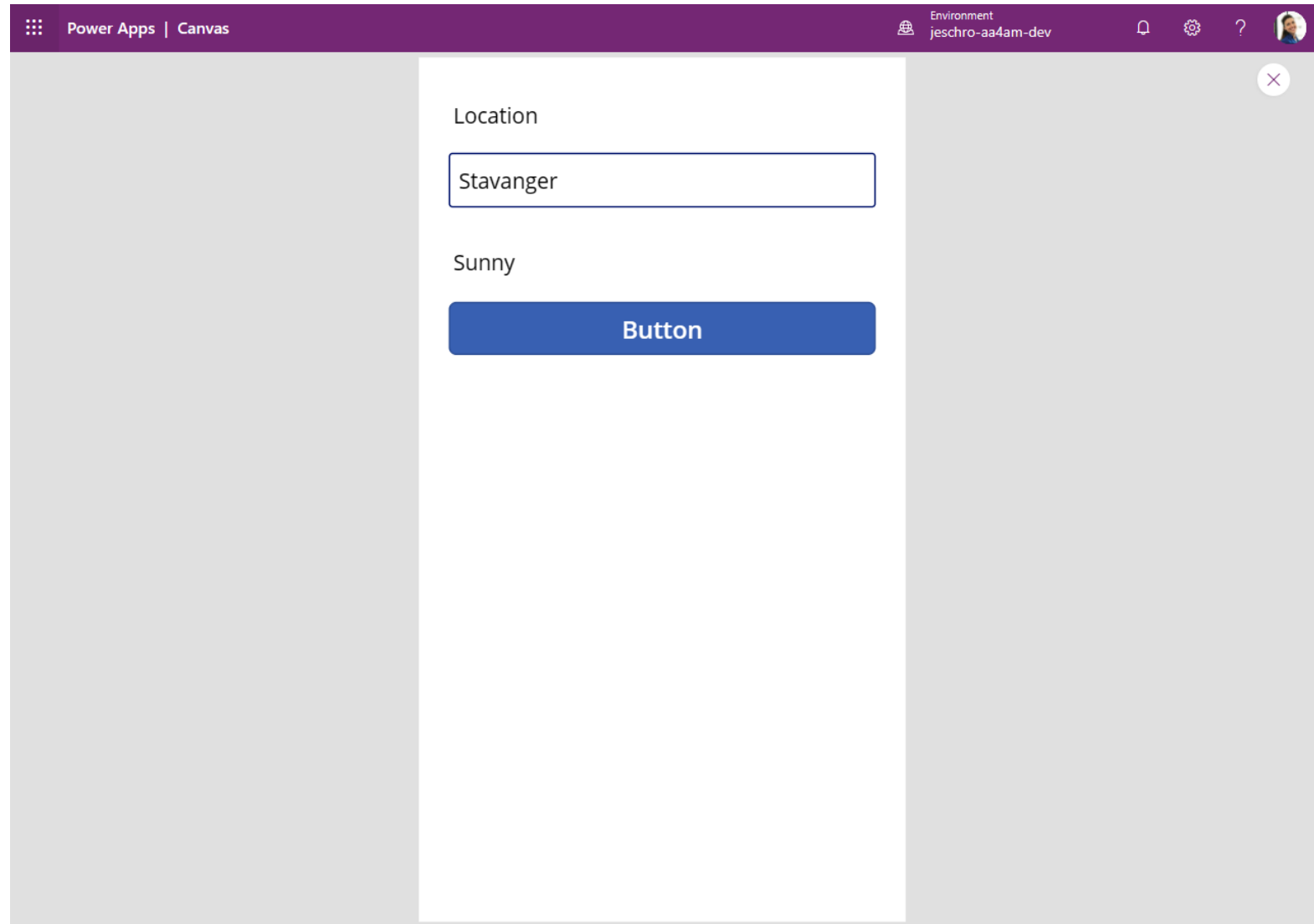
The label will show the value of the forecastResult variable



# Step 40. Test your app

Play your app by clicking the play button (▶) in the top right corner or pressing F5

Enter a location and press the button

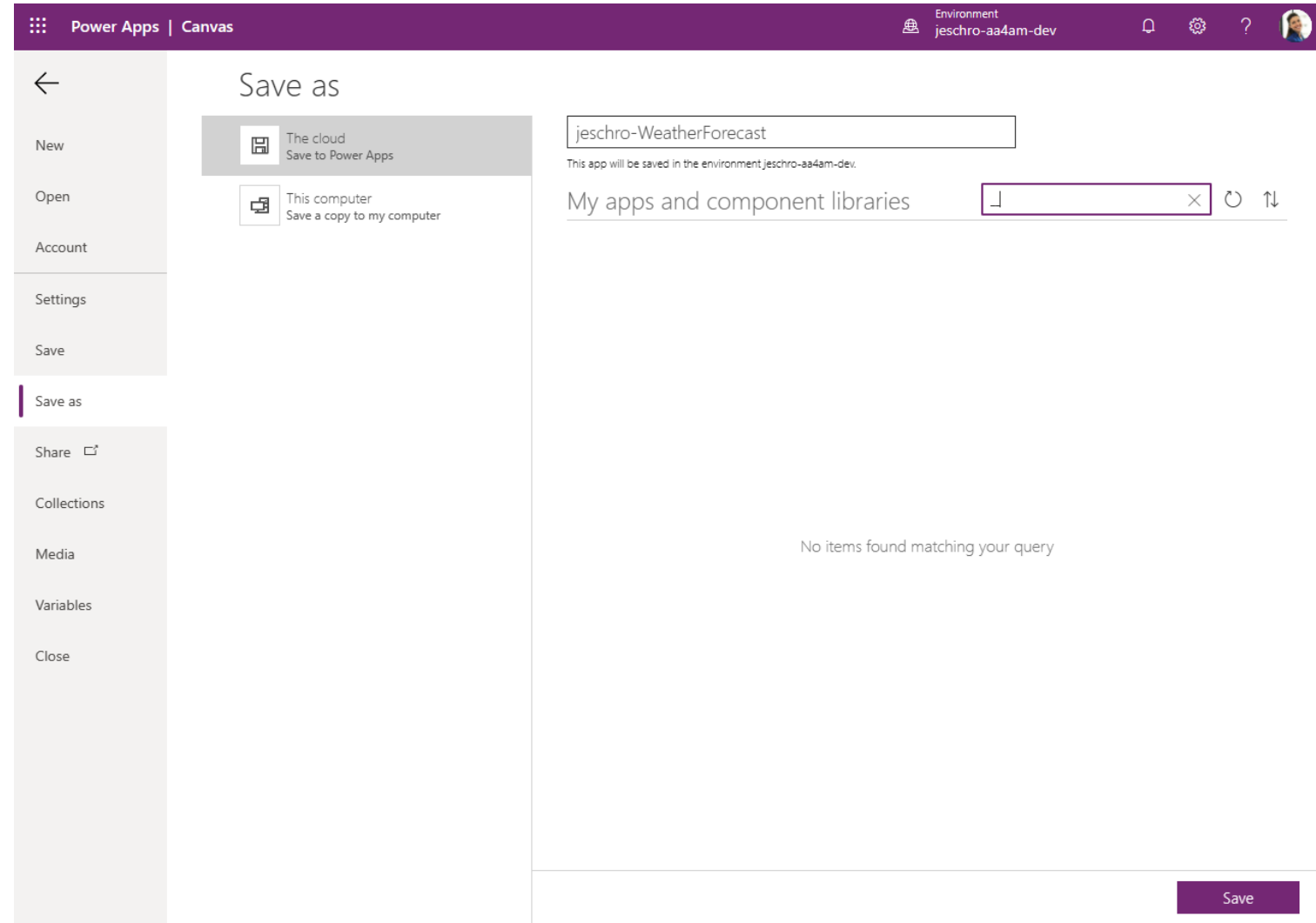


# Step 41. Save your app

Close the app player in top right corner.

Click **File => Save** to save your app

Your app calling an Azure Function via custom connector is now ready to be used via mobile phone, browser, teams client or embedded in other web technologies.

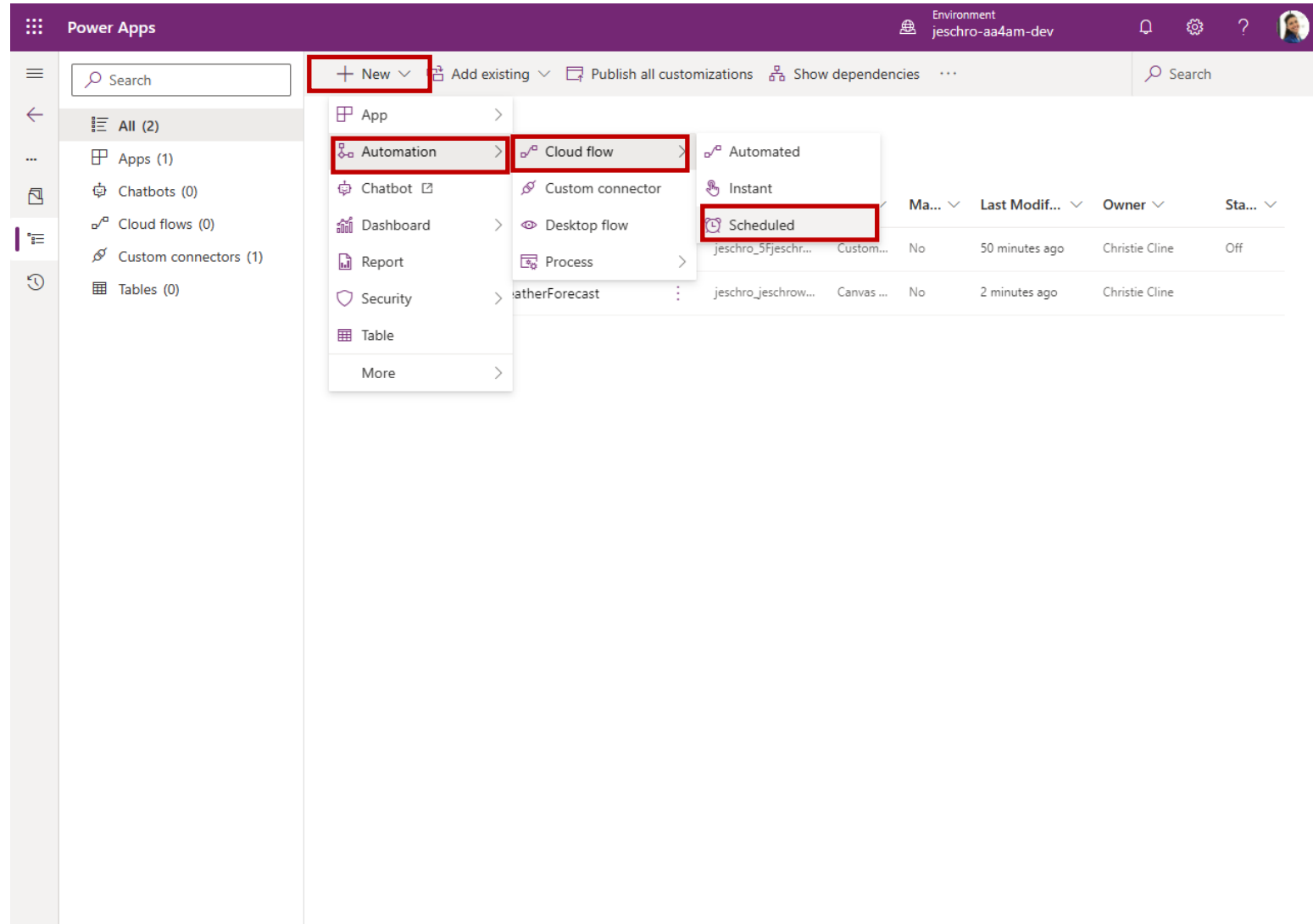


# Using a Power Platform custom connector in a Power Automate Flow

# Step 42. Add a Scheduled Cloud flow to your solution

Navigate to your solution

Click **+New => Cloud flow => Scheduled**



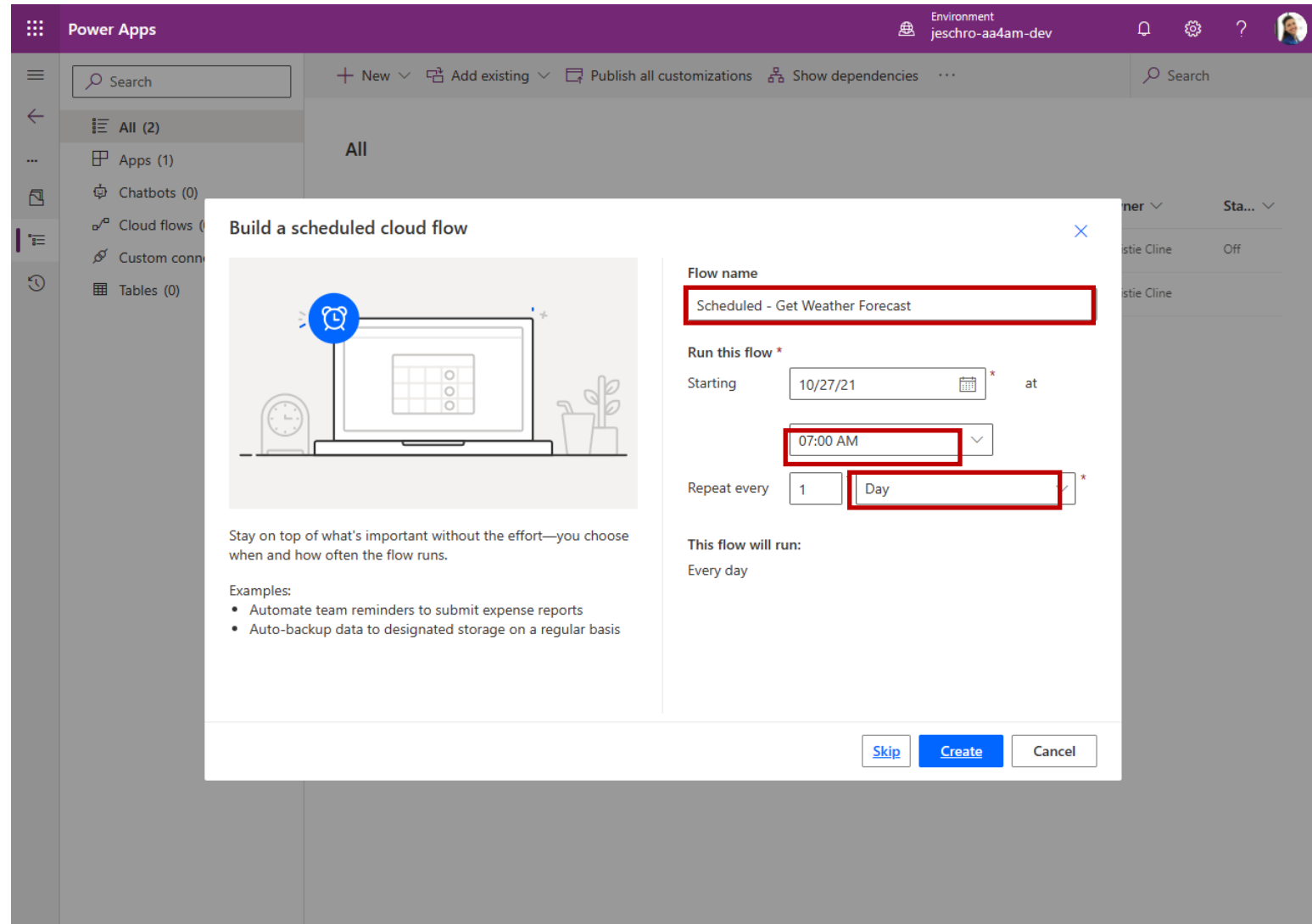


# Step 43. Configure Schedule and create cloud flow

Create your cloud flow with the following properties:

- Flow name: Scheduled – Get Weather Forecast
- Starting: [today] at 7:00 AM
- Repeat every: 1 Day

Click **Create**



The screenshot shows the 'Build a scheduled cloud flow' dialog box in the Power Apps environment. The dialog is titled 'Build a scheduled cloud flow' and features a blue clock icon. The 'Flow name' field is set to 'Scheduled - Get Weather Forecast'. The 'Run this flow' section shows the flow starting on '10/27/21' at '07:00 AM'. The 'Repeat every' section is set to '1 Day'. The 'This flow will run:' section indicates 'Every day'. At the bottom, there are three buttons: 'Skip', 'Create', and 'Cancel'. The 'Create' button is highlighted in blue.

Power Apps

Environment: jeschro-aa4am-dev

Search

All (2)

Apps (1)

Chatbots (0)

Cloud flows (0)

Custom connections (0)

Tables (0)

All

Build a scheduled cloud flow

Flow name: Scheduled - Get Weather Forecast

Run this flow \*

Starting: 10/27/21 at 07:00 AM

Repeat every: 1 Day

This flow will run: Every day

Skip Create Cancel

# Step 44. Add a new step to your flow

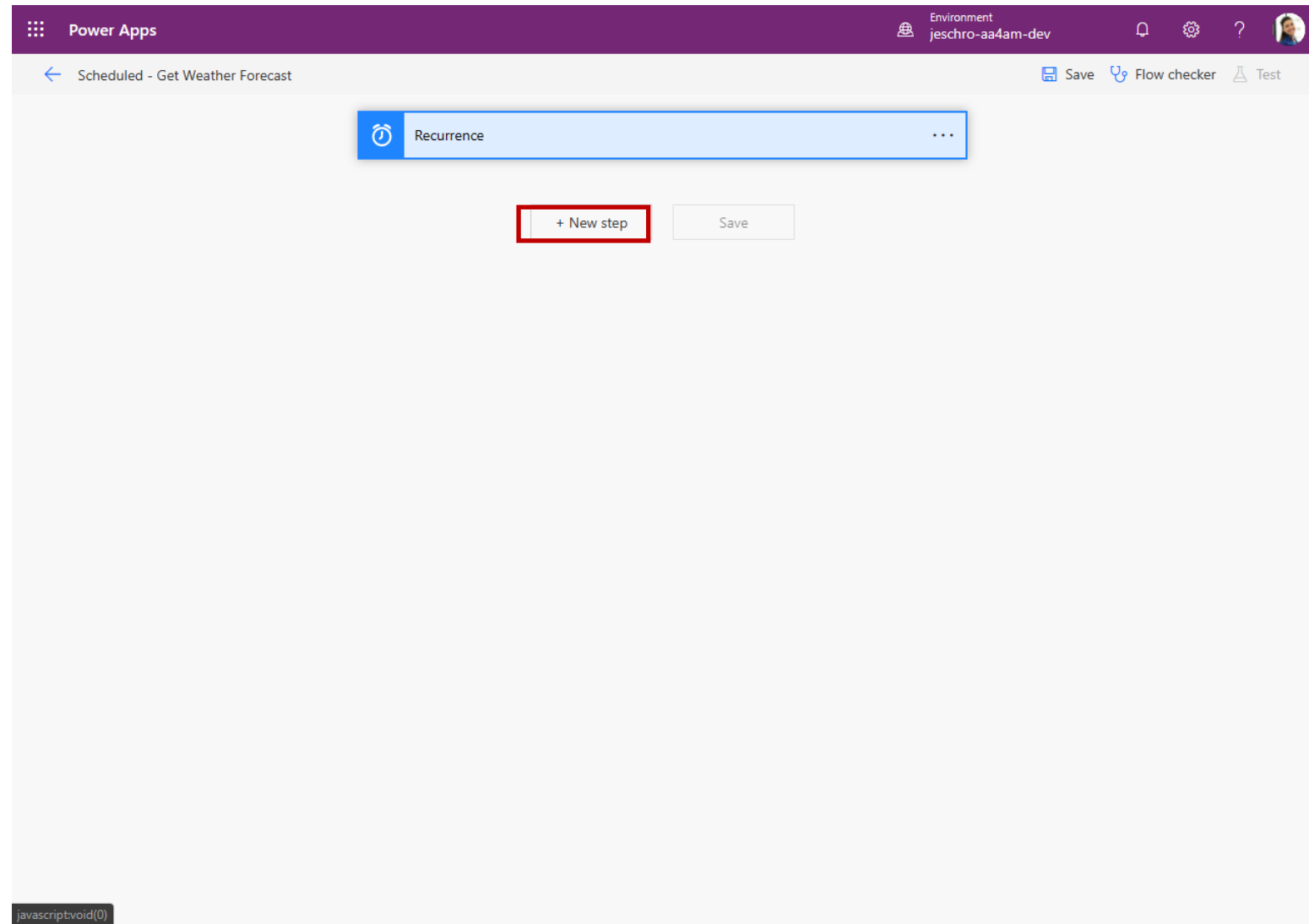
Your flow has been created with only a trigger.

The trigger is what starts the flow.

As defined when the flow was created the trigger is a Recurring trigger with a schedule.

We now must add at least one action to the flow.

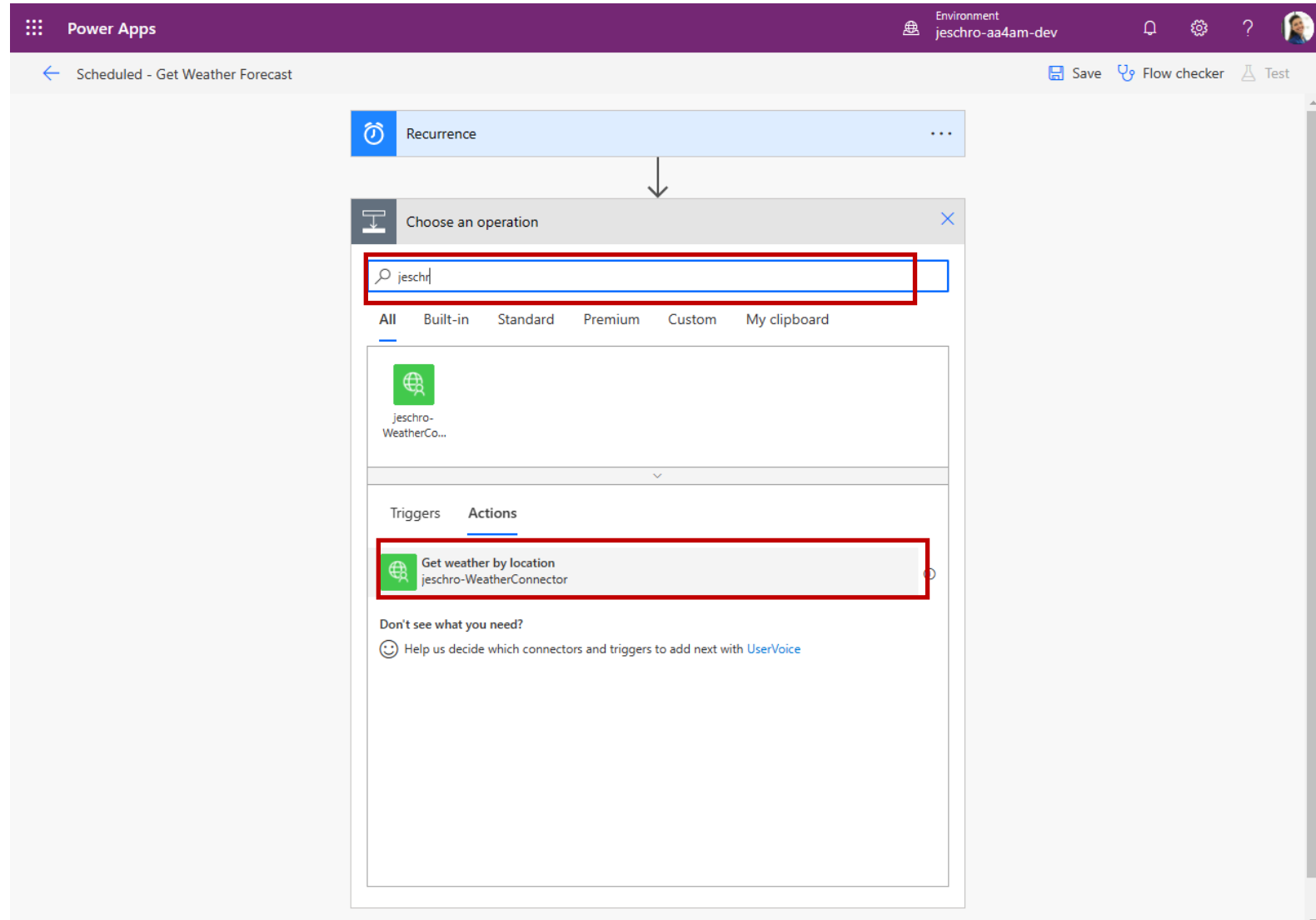
Click **+New step**



# Step 45. Add the Get weather by location action

Search for your alias to find your custom connector

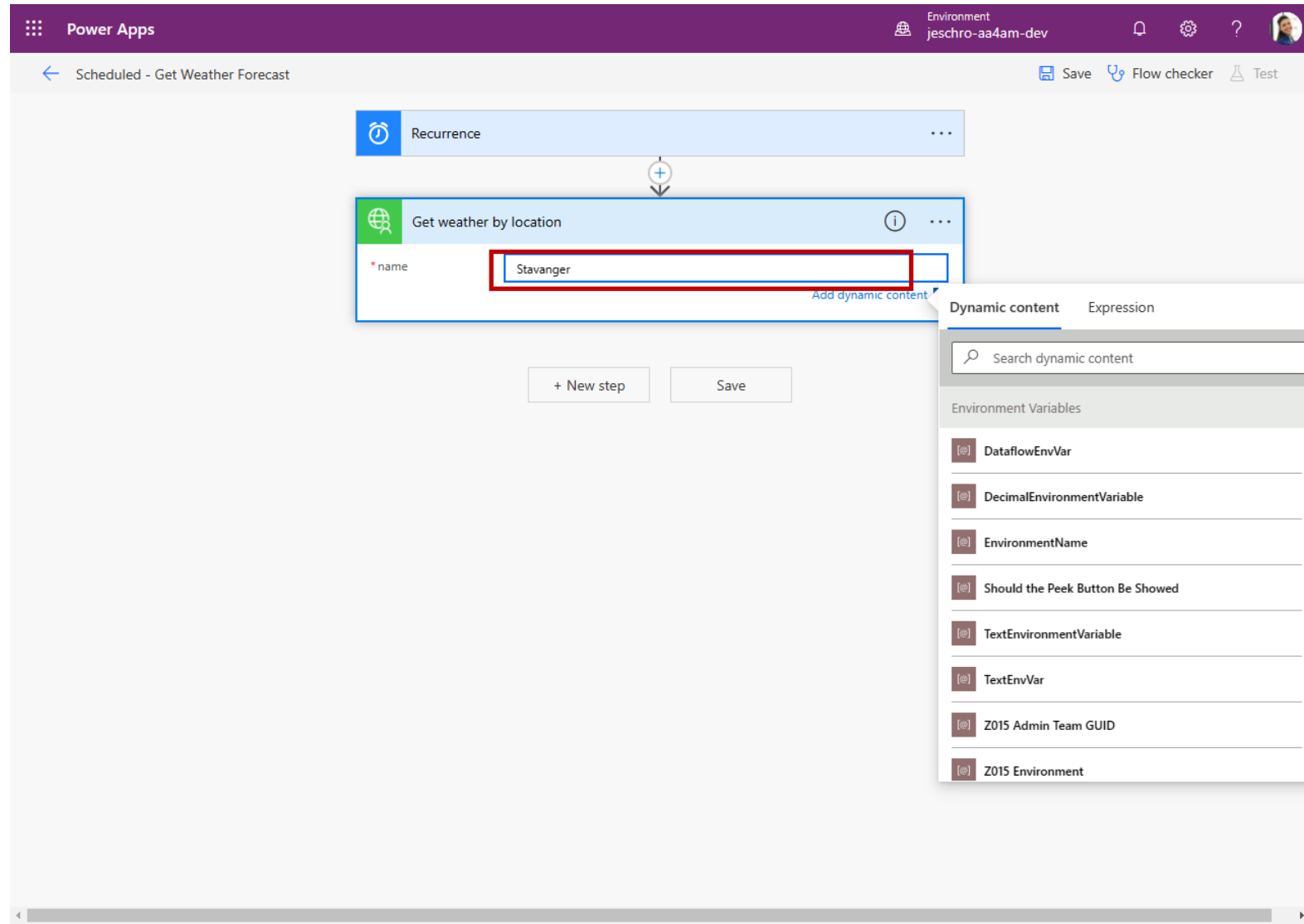
Click the **Get weather by location** action



# Step 46. Configure the Get weather by location Action

For the purpose of the lab we will hardcode in a location value.

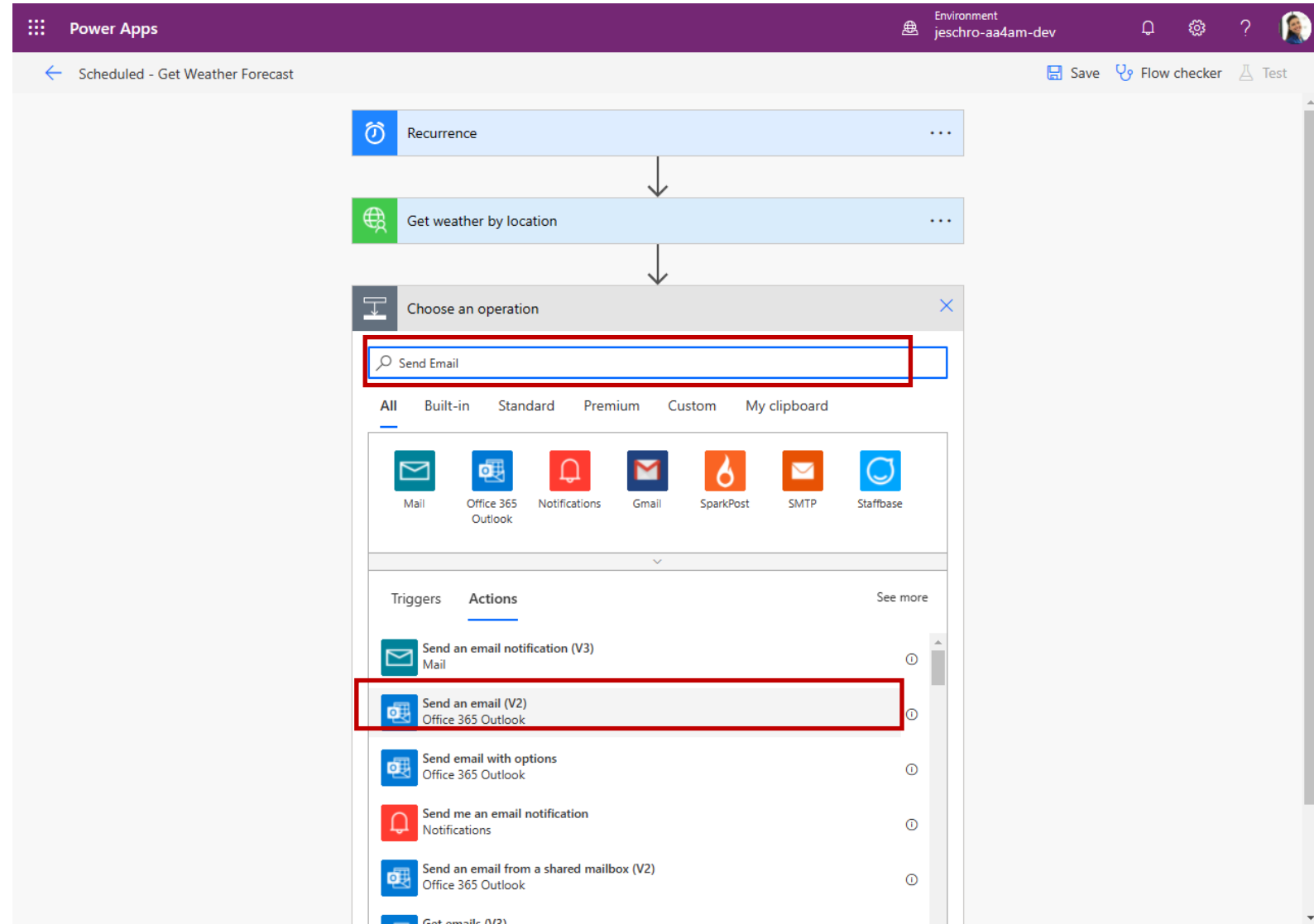
Enter a location (Bergen, Oslo, Stavanger) in the **name** property.



# Step 47. Add the Send an email (v2) action

Lets complete the cloud flow by sending the forecast via email

Click + **New step** and select the Send an email (v2) action



# Step 48. Configure the Send an Email (v2) action

Configure the following properties of the Send an email (v2) action

- To: your email address
- Subject : "Weather forecast for Stavanger : "

While in the Subject property scroll down in the Dynamics content pop up and click **forecast**.

This will add a reference to forecast property in the output of the Get weather by location action

The screenshot shows the Power Apps interface for configuring a 'Send an email (V2)' action. The 'To' field is set to 'Jens Schrøder'. The 'Subject' field is 'Weather forecast for Stavanger : ' and is highlighted with a red box. The 'Body' field is empty. A 'Dynamic content' panel is open on the right, showing a list of properties. The 'forecast' property is highlighted with a red box.

Power Apps Environment: jeschro-aa4am-dev

Scheduled - Get Weather Forecast

Recurrence

Get weather by location

Send an email (V2)

To: Jens Schrøder

Subject: Weather forecast for Stavanger :

Body: Specify the body of the mail

Dynamic content

Search dynamic content

TextEnvVar

2015 Admin Team GUID

2015 Environment

2015 Plant Editor Team GUID

2015\_Web\_Prod\_Instance\_Id

2015\_Web\_Test\_Instance\_Id

Get weather by location

locationName

forecast

# Step 49. Configure the Send an Email (v2) action continued

Copy and paste the content from the Subject property to the Body property

The screenshot shows the Power Apps interface for configuring a 'Send an email (V2)' action within a flow named 'Scheduled - Get Weather Forecast'. The flow steps are: 'Recurrence' followed by 'Get weather by location', which then leads to the 'Send an email (V2)' action. The email configuration is as follows:

- To:** Jens Schröder
- Subject:** Weather forecast for Stavanger : forecast
- Body:** Weather forecast for Stavanger : forecast

The 'Dynamic content' panel is open, displaying a search bar and a list of environment variables:

- Environment Variables
- DataflowEnvVar
- EnvironmentName
- Should the Peek Button Be Shown
- TextEnvironmentVariable
- TextEnvVar
- Z015 Admin Team GUID

# Step 50. Save and Test

Click **Test** in the top right corner to test the cloud flow.

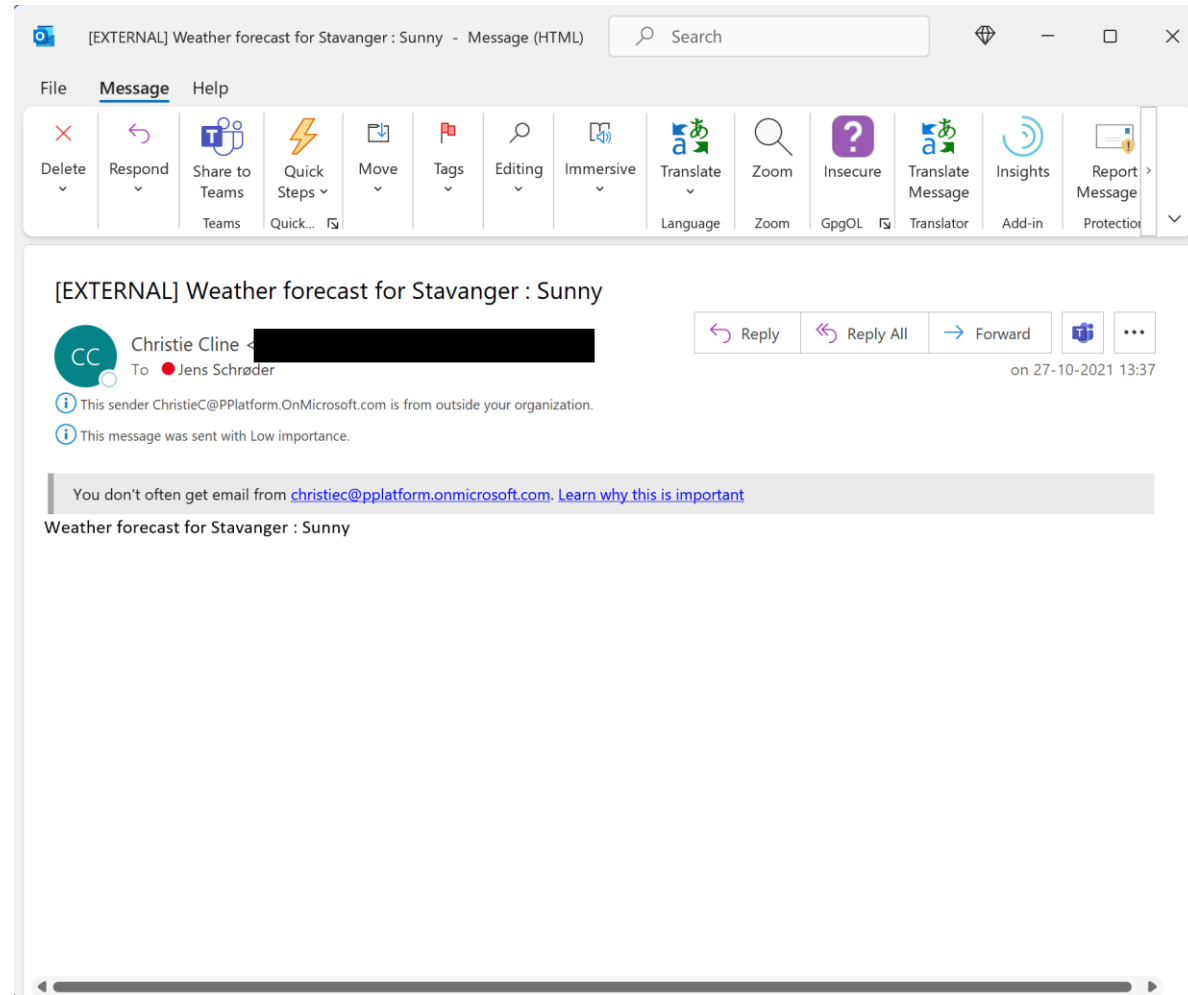
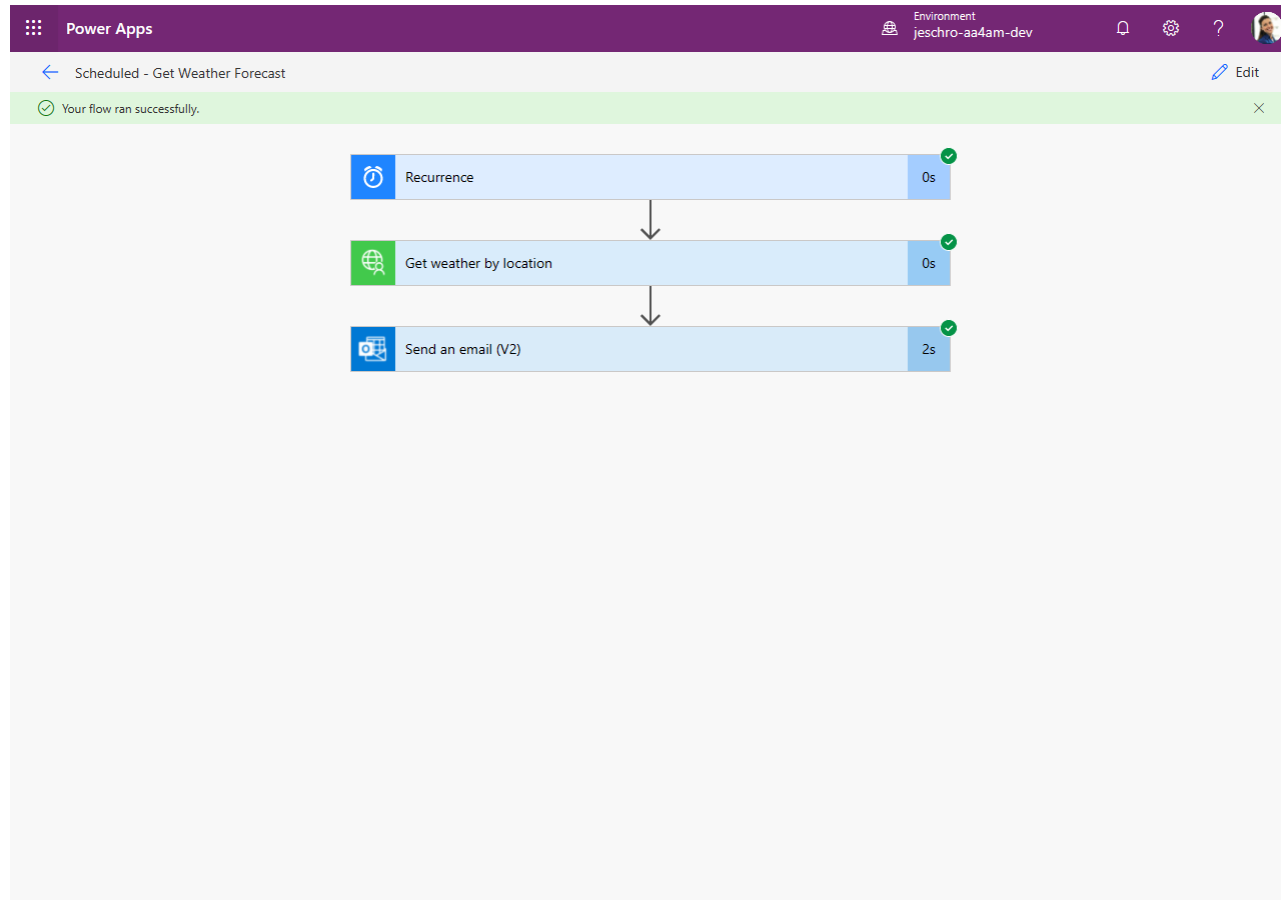
Select **Manually** and click **Save & Test**

Then click **Run flow**

The screenshot displays the Microsoft Power Apps interface for testing a cloud flow. The top bar shows 'Power Apps' and the environment 'jeschro-aa4am-dev'. The flow being tested is 'Scheduled - Get Weather Forecast'. The flow steps are: 'Recurrence' (blue step), 'Get weather by location' (green step), and 'Send an email (V2)' (blue step). The 'Send an email (V2)' step is expanded, showing fields for 'To' (Jens Schrøder), 'Subject' (Weather forecast for Stavanger : forecast), and 'Body' (Weather forecast for Stavanger : forecast). The 'Test Flow' dialog box is open on the right, with the 'Manually' radio button selected (highlighted with a red box). At the bottom right, the 'Save & Test' button is highlighted with a red box, next to a 'Cancel' button.



# Step 51. Verify the run is successful



Create a custom connector for Azure AD  
protected Azure Functions | Microsoft Docs

How To: Use Swagger in Azure Functions -  
Cloudkasten