# Custom Connector Lab

Power Platform Workshop

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

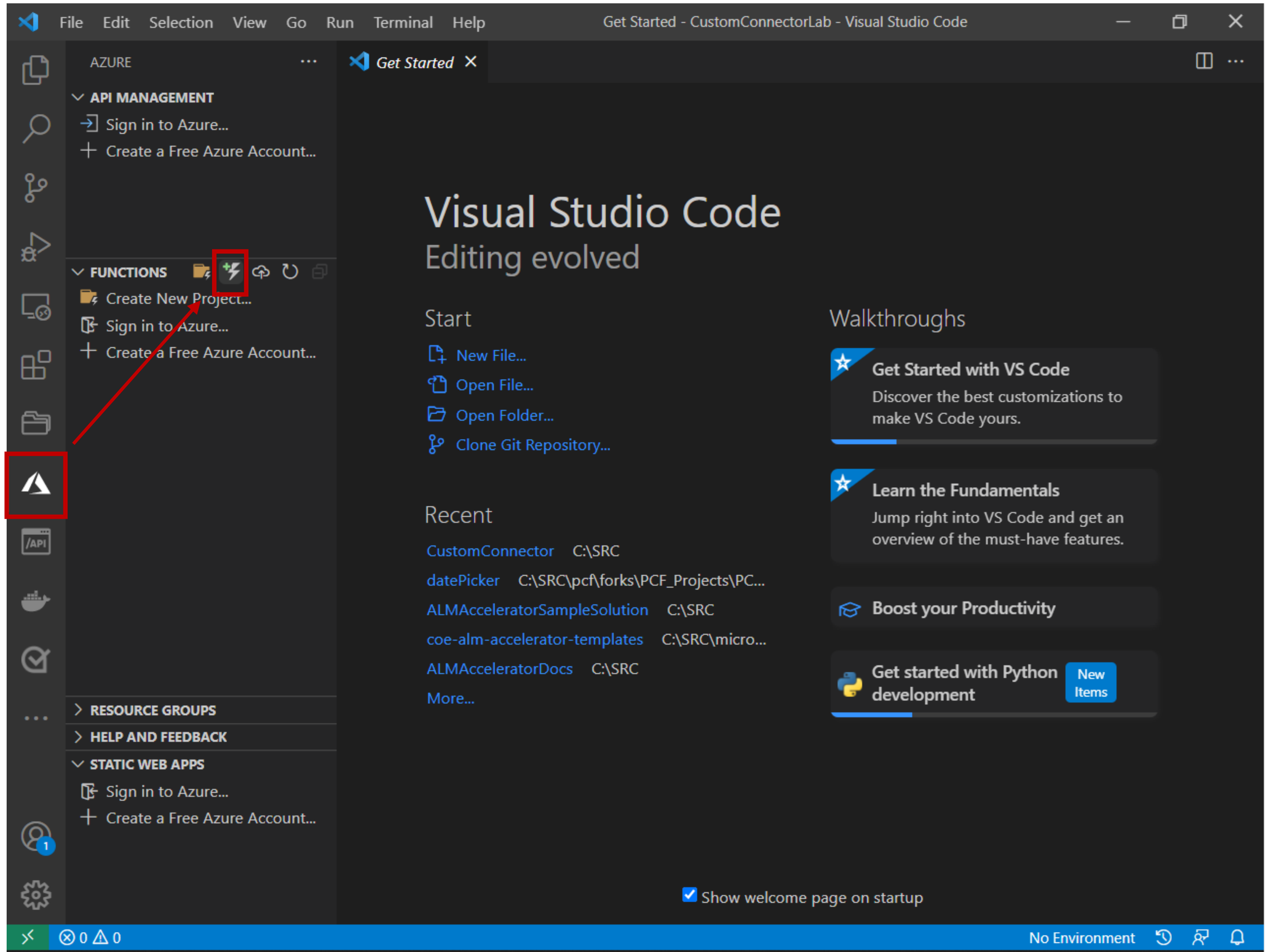
Select the Azure Functions extension for Visual Studio Code and add a new function using the lighting bolt icon.

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*

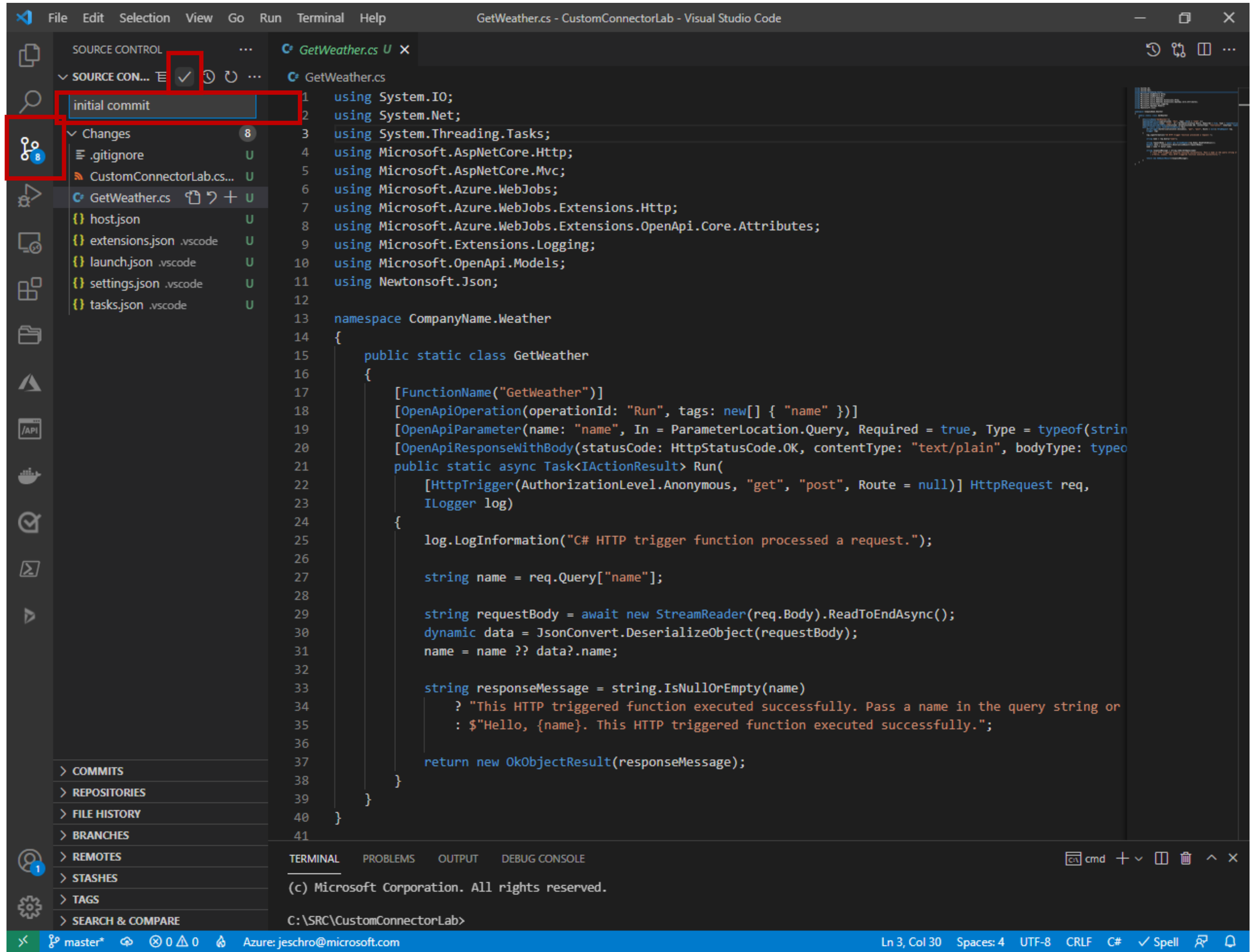
*For the purpose of this lab we will use anonymous access rights.
However, it is recommended to always secure your functions


File Edit Selection View Go Run Terminal Help
Get Started - CustomConnectorLab - Visual Studio Code
AZURE
API MANAGEMENT
Sign in to Azure...
Create a Free Azure Account...
FUNCTIONS
Create New Project...
Sign in to Azure...
Create a Free Azure Account...
RESOURCE GROUPS
HELP AND FEEDBACK
STATIC WEB APPS
Sign in to Azure...
Create a Free Azure Account...
Get Started
Visual Studio Code
Editing evolved
Start
New File...
Open File...
Open Folder...
Clone Git Repository...
Recent
CustomConnector C:\SRC
datePicker C:\SRC\pcf\forks\PCF_Projects\PC...
ALMAcceleratorSampleSolution C:\SRC
coe-alm-accelerator-templates C:\SRC\micro...
ALMAcceleratorDocs C:\SRC
More...
Walkthroughs
Get Started with VS Code
Discover the best customizations to make VS Code yours.
Learn the Fundamentals
Jump right into VS Code and get an overview of the must-have features.
Boost your Productivity
Get started with Python development
New Items
Show welcome page on startup
No Environment

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

Select the GitLens extension

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



```
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;

namespace CompanyName.Weather
{
    public static class GetWeather
    {
        [FunctionName("GetWeather")]
        [OpenApiOperation(operationId: "Run", tags: new[] { "name" })]
        [OpenApiParameter(name: "name", In = ParameterLocation.Query, Required = true, Type = typeof(strin
        [OpenApiResponseWithBody(statusCode: HttpStatusCode.OK, contentType: "text/plain", bodyType: typeo
        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;

            string responseMessage = string.IsNullOrEmpty(name)
                ? "This HTTP triggered function executed successfully. Pass a name in the query string or
                : $"Hello, {name}. This HTTP triggered function executed successfully.";

            return new OkObjectResult(responseMessage);
        }
    }
}
```

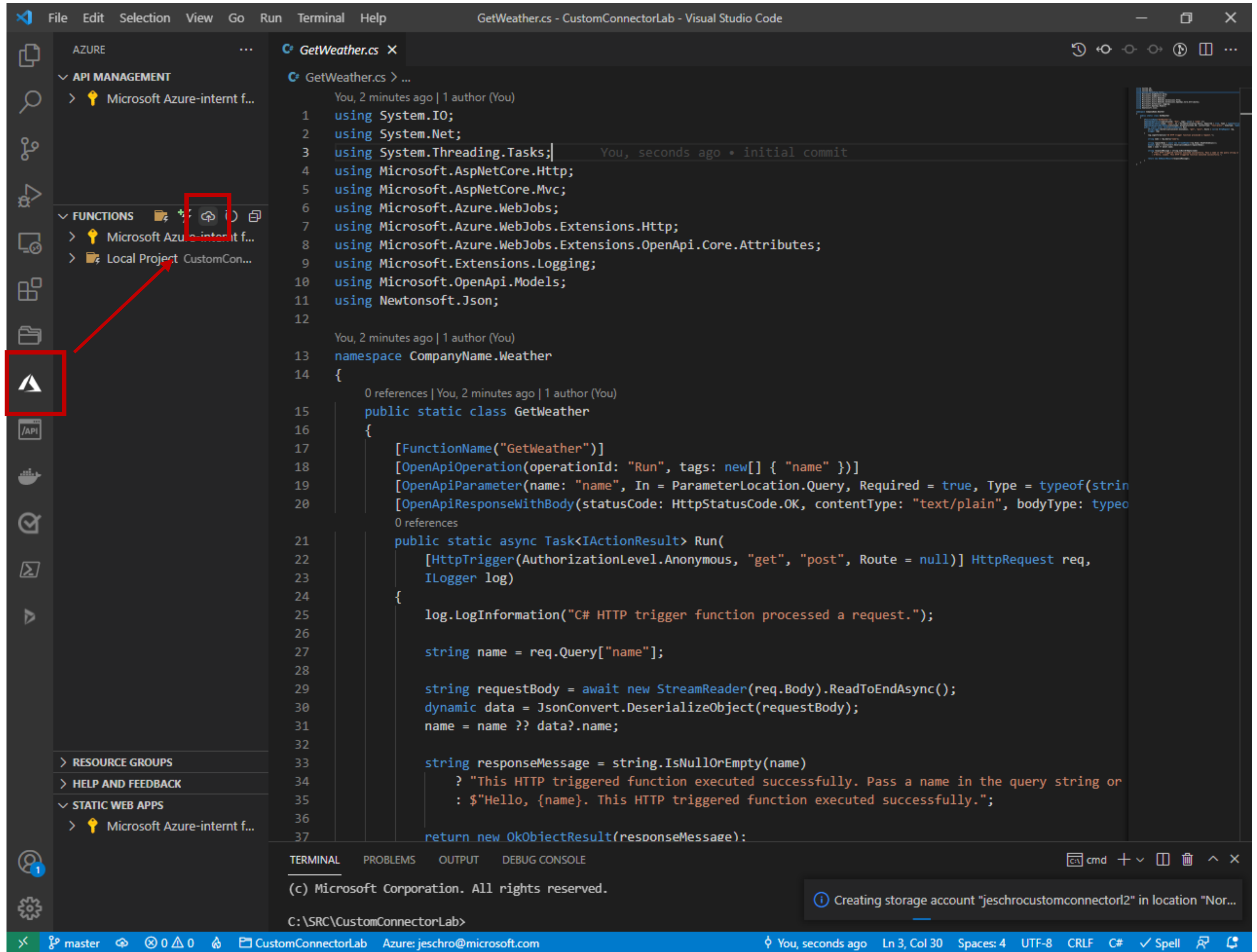
```
(c) Microsoft Corporation. All rights reserved.

C:\SRC\CustomConnectorLab>
```

# 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)



```
You, 2 minutes ago | 1 author (You)
using System.IO;
using System.Net;
using System.Threading.Tasks;        You, seconds ago • initial commit
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;

You, 2 minutes ago | 1 author (You)
namespace CompanyName.Weather
{
    0 references | You, 2 minutes ago | 1 author (You)
    public static class GetWeather
    {
        [FunctionName("GetWeather")]
        [OpenApiOperation(operationId: "Run", tags: new[] { "name" })]
        [OpenApiParameter(name: "name", In = ParameterLocation.Query, Required = true, Type = typeof(strin
        [OpenApiResponseWithBody(statusCode: HttpStatusCode.OK, contentType: "text/plain", bodyType: typeo
        0 references
        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;

            string responseMessage = string.IsNullOrEmpty(name)
                ? "This HTTP triggered function executed successfully. Pass a name in the query string or
                : $"Hello, {name}. This HTTP triggered function executed successfully.";

            return new OkObjectResult(responseMessage);
```

```
(c) Microsoft Corporation. All rights reserved.

C:\SRC\CustomConnectorLab>
```

Creating storage account "jeschrocustomconnectorl2" in location "Nor...

## 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:
```

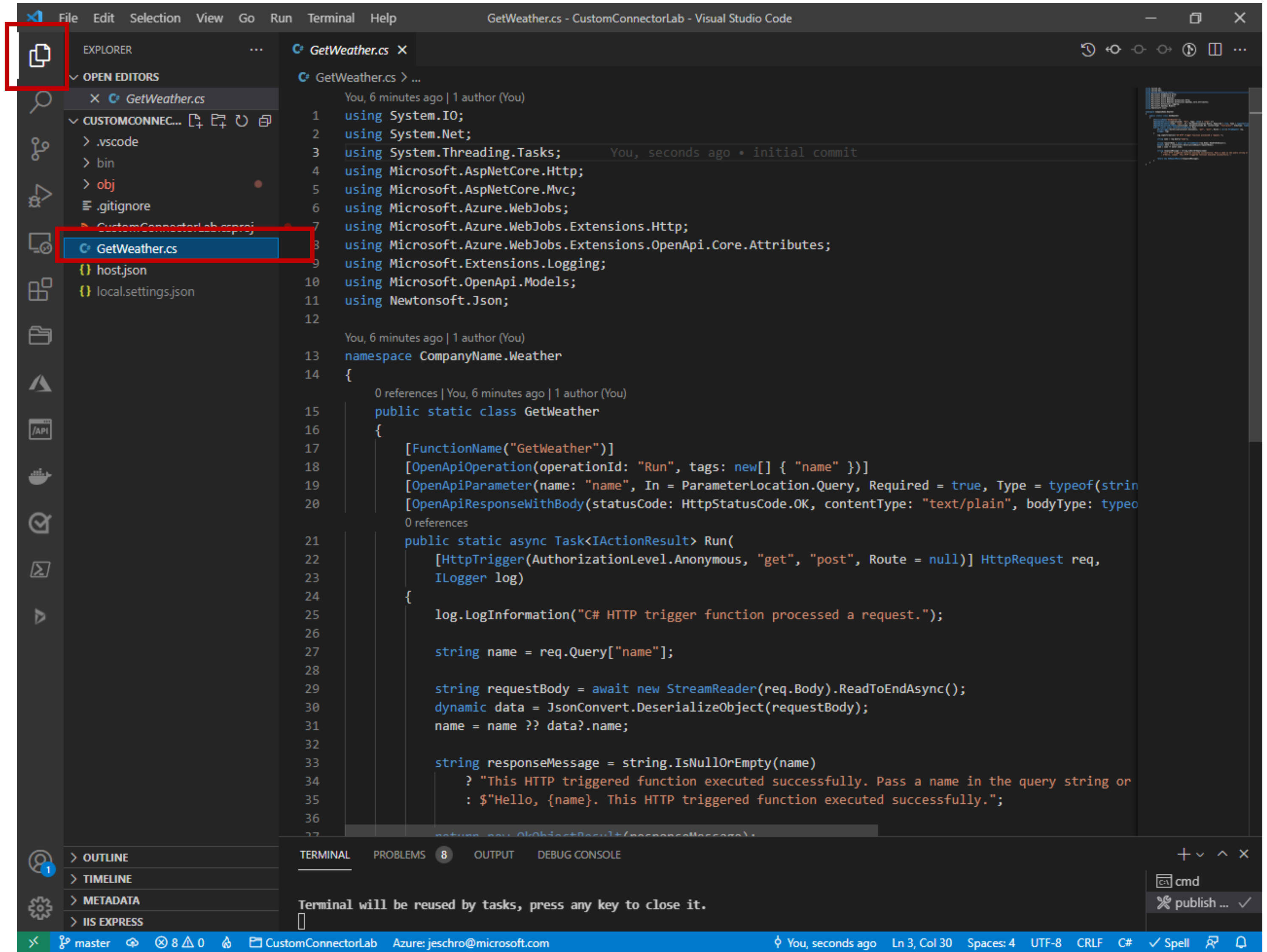
*“Hello, Jens. This HTTP triggered function executed successfully.”*

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



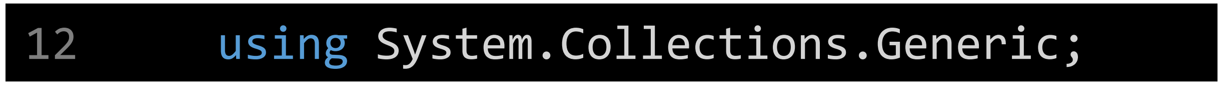
```
using System.IO;
using System.Net;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Azure.WebJobs;
using Microsoft.Azure.WebJobs.Extensions.Http;
using Microsoft.Azure.WebJobs.Extensions.OpenApi.Core.Attributes;
using Microsoft.Extensions.Logging;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

namespace CompanyName.Weather
{
    public static class GetWeather
    {
        [FunctionName("GetWeather")]
        [OpenApiOperation(operationId: "Run", tags: new[] { "name" })]
        [OpenApiParameter(name: "name", In = ParameterLocation.Query, Required = true, Type = typeof(strin
        [OpenApiResponseWithBody(statusCode: HttpStatusCode.OK, contentType: "text/plain", bodyType: typeo
        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;

            string responseMessage = string.IsNullOrEmpty(name)
                ? "This HTTP triggered function executed successfully. Pass a name in the query string or
                : $"Hello, {name}. This HTTP triggered function executed successfully.";
```

```
Terminal will be reused by tasks, press any key to close it.
```

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

Add the following line of code at the end of the exisiting using statements in the top of you function code

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



```
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 **}** )

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

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

# 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

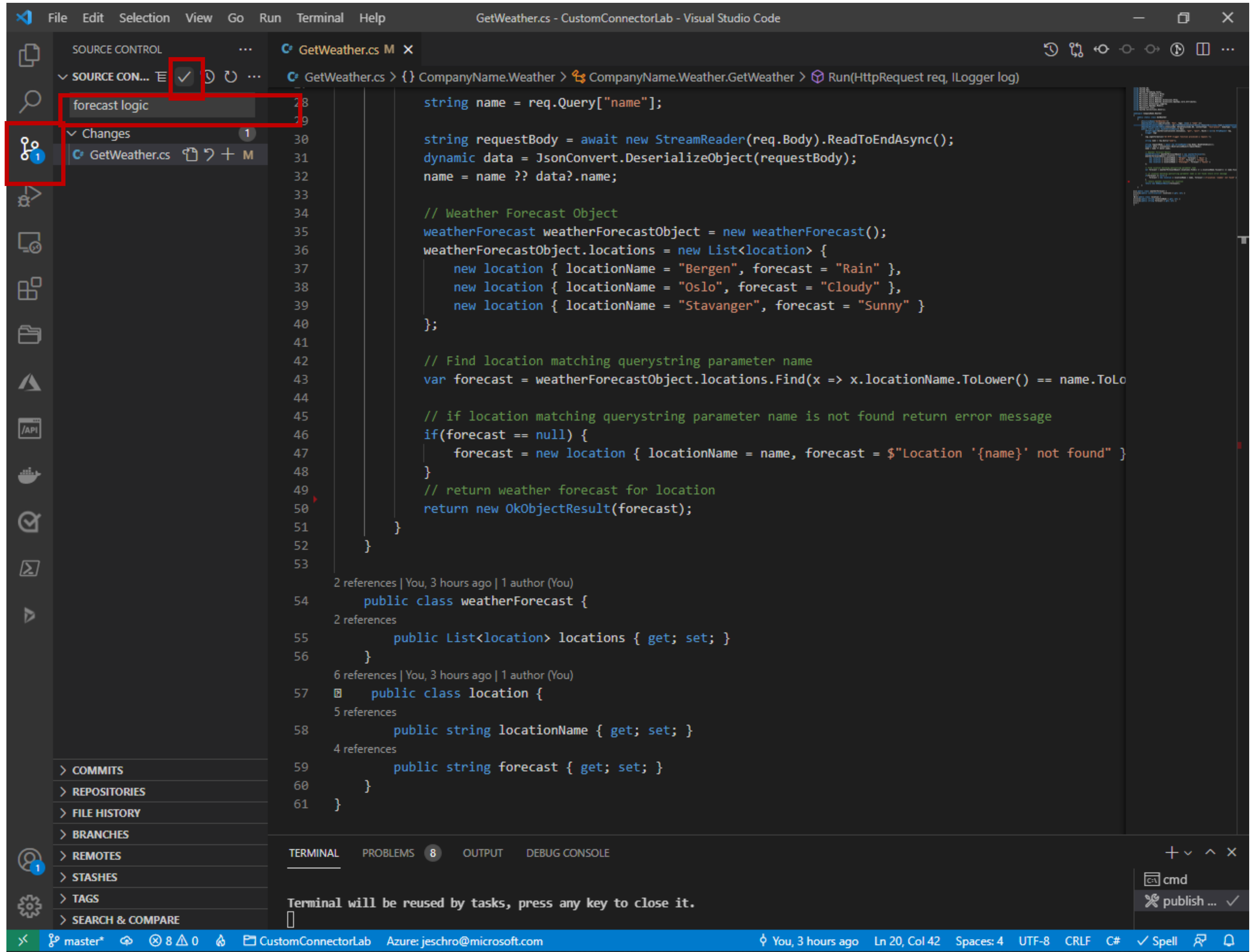
```
using System.IO;
using System.Net;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Azure.WebJobs;
using Microsoft.Azure.WebJobs.Extensions.Http;
using Microsoft.Azure.WebJobs.Extensions.OpenApi.Core.Attributes;
using Microsoft.Extensions.Logging;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

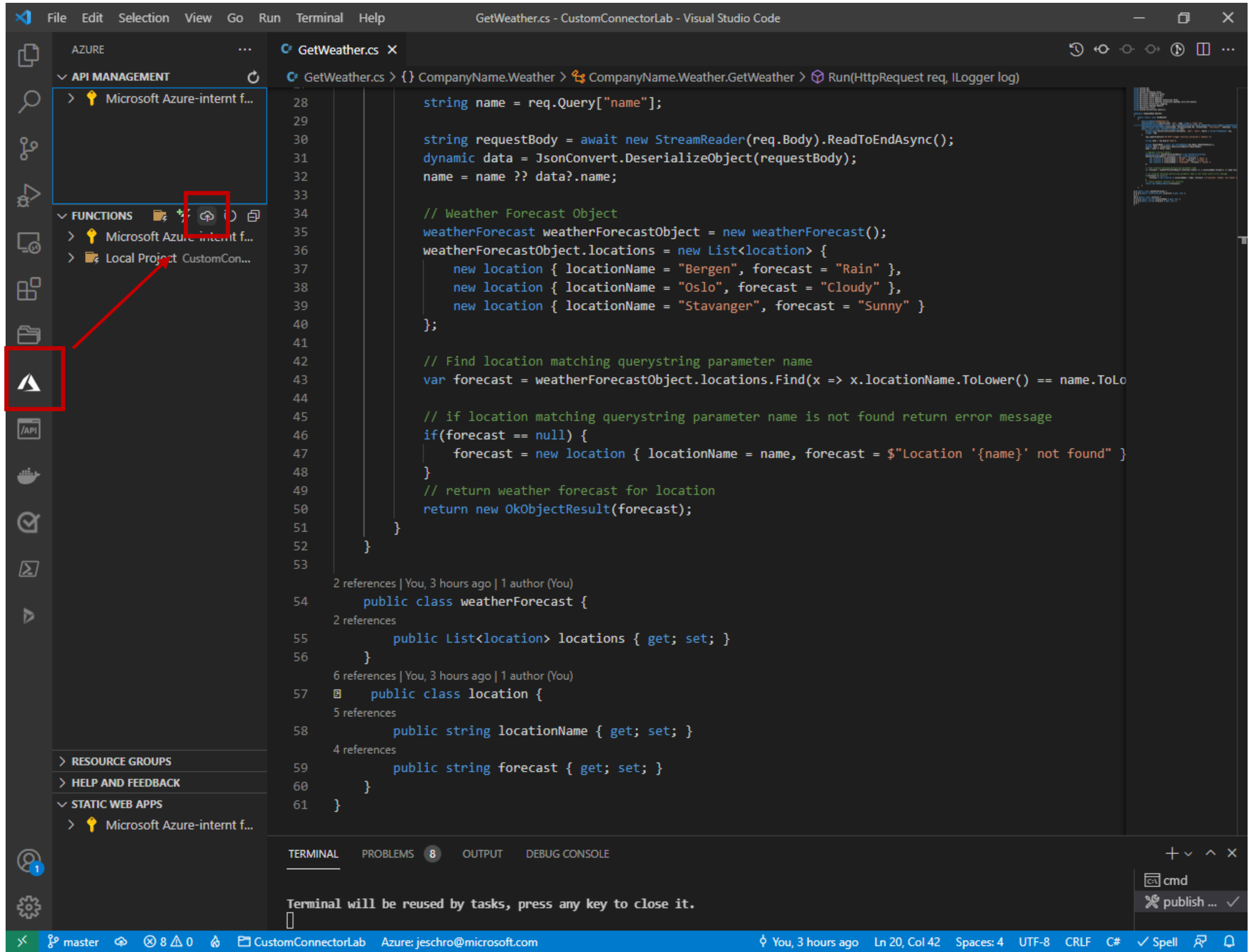


```
            string name = req.Query["name"];

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

            // 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.ToLo

            // 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);
        }
    }

    2 references | You, 3 hours ago | 1 author (You)
    public class weatherForecast {
    2 references
        public List<location> locations { get; set; }
    }
    6 references | You, 3 hours ago | 1 author (You)
    public class location {
    5 references
        public string locationName { get; set; }
    4 references
        public string forecast { get; set; }
    }
}
```

```
Terminal will be reused by tasks, press any key to close it.
```

# Step 11. Deploy these updates:

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


File Edit Selection View Go Run Terminal Help
GetWeather.cs - CustomConnectorLab - Visual Studio Code
AZURE
API MANAGEMENT
Microsoft Azure-internt f...
FUNCTIONS
Microsoft Azure-internt f...
Local Project CustomCon...
RESOURCE GROUPS
HELP AND FEEDBACK
STATIC WEB APPS
Microsoft Azure-internt f...
GetWeather.cs
GetWeather.cs > {} CompanyName.Weather > CompanyName.Weather.GetWeather > Run(HttpRequest req, ILogger log)
TERMINAL PROBLEMS 8 OUTPUT DEBUG CONSOLE
cmd
publish ...
master ⊗ 8 ⚠ 0 CustomConnectorLab Azure: jeschro@microsoft.com You, 3 hours ago Ln 20, Col 42 Spaces: 4 UTF-8 CRLF C# ✓ Spell

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

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

            // 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.ToLo

            // 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);
        }
    }

    2 references | You, 3 hours ago | 1 author (You)
    public class weatherForecast {
        2 references
        public List<location> locations { get; set; }
    }
    6 references | You, 3 hours ago | 1 author (You)
    public class location {
        5 references
        public string locationName { get; set; }
        4 references
        public string forecast { get; set; }
    }
}
```

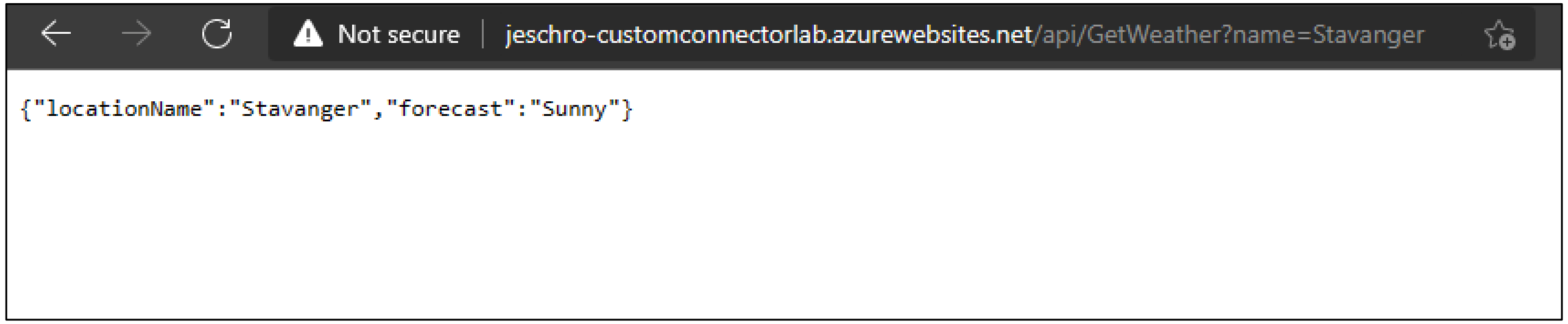
```
Terminal will be reused by tasks, press any key to close it.
```

# 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!***

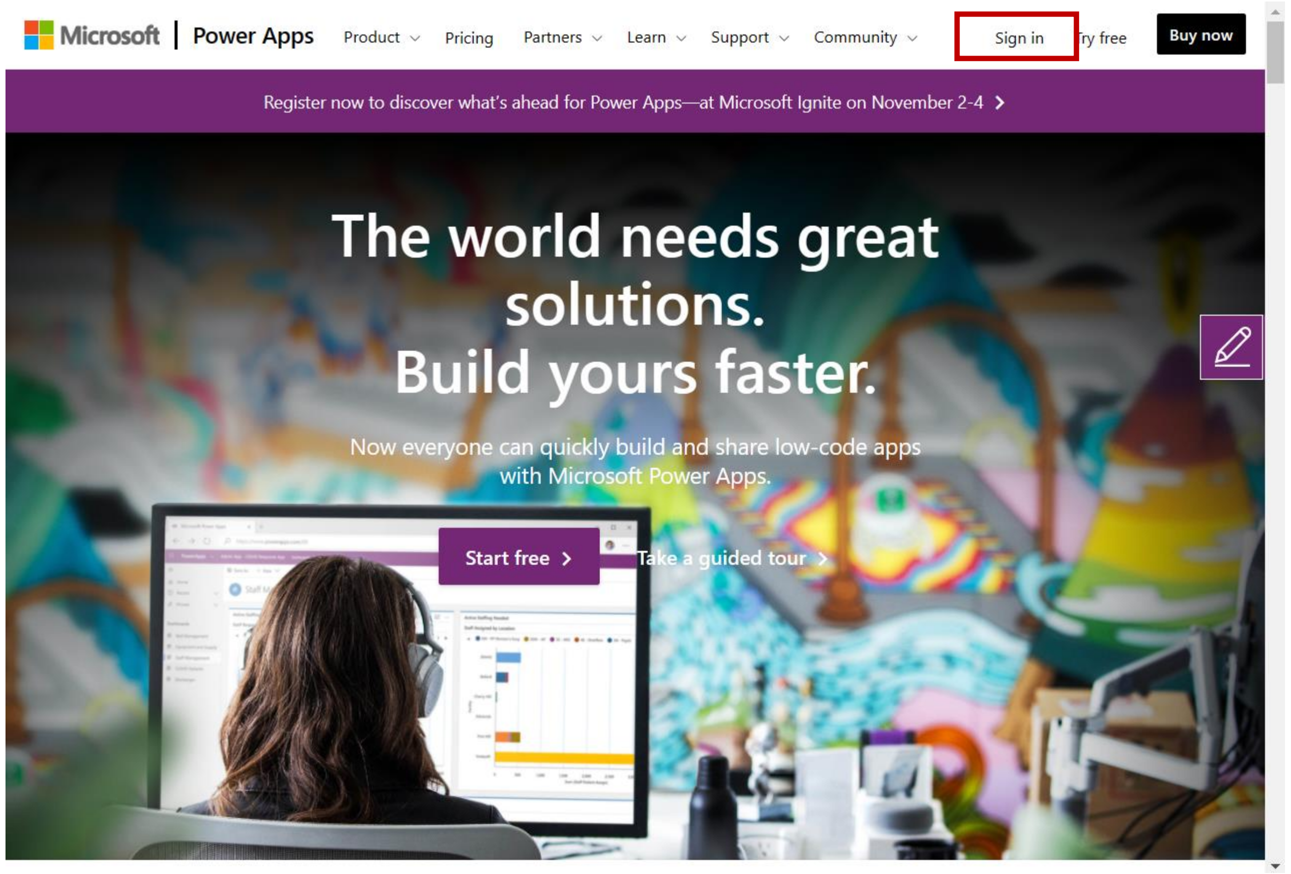

Not secure
jeschro-customconnectorlab.azurewebsites.net/api/GetWeather?name=Stavanger

```
{"locationName":"Stavanger","forecast":"Sunny"}
```

## 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



Register now to discover what's ahead for Power Apps—at Microsoft Ignite on November 2-4

The world needs great solutions.
Build yours faster.

Now everyone can quickly build and share low-code apps with Microsoft Power Apps.

# Step 14. Create a solution

Once signed in click **Solutions**.

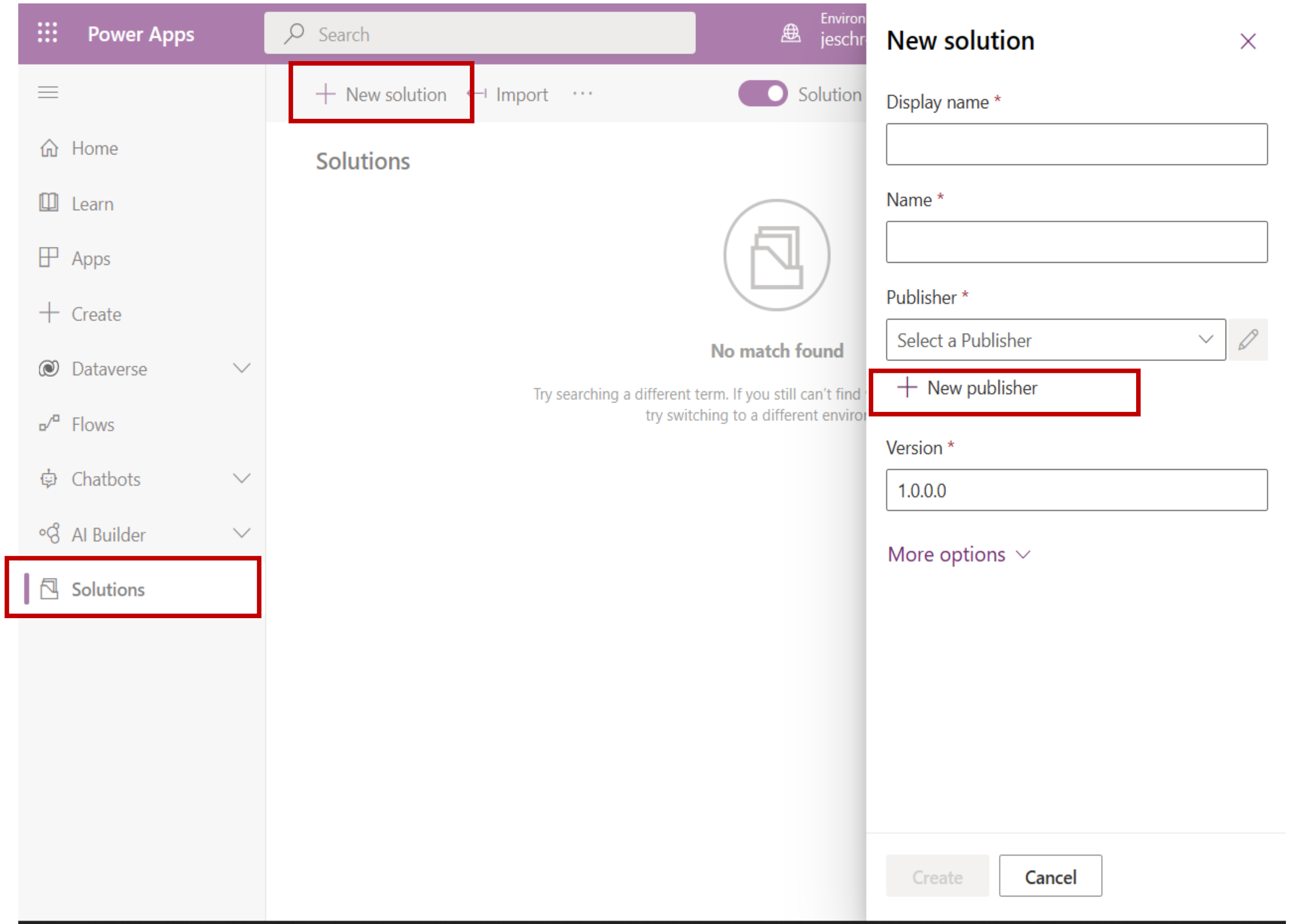
Then click **+ New solution**

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.

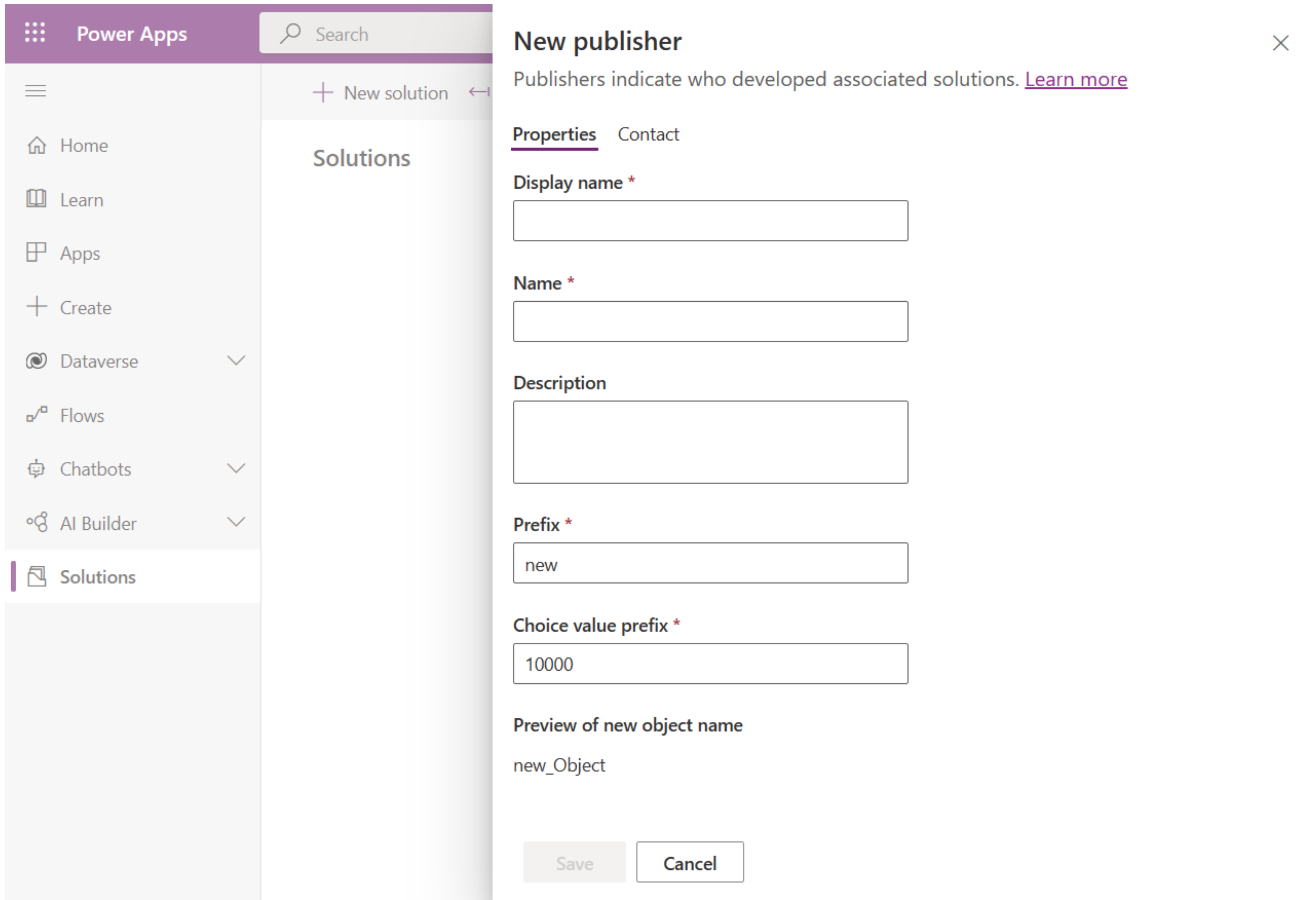


# Step 15. Create a Publisher

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


New publisher

Publishers indicate who developed associated solutions. Learn more

Display name *

Name *

Description

Prefix *: new

Choice value prefix *: 10000

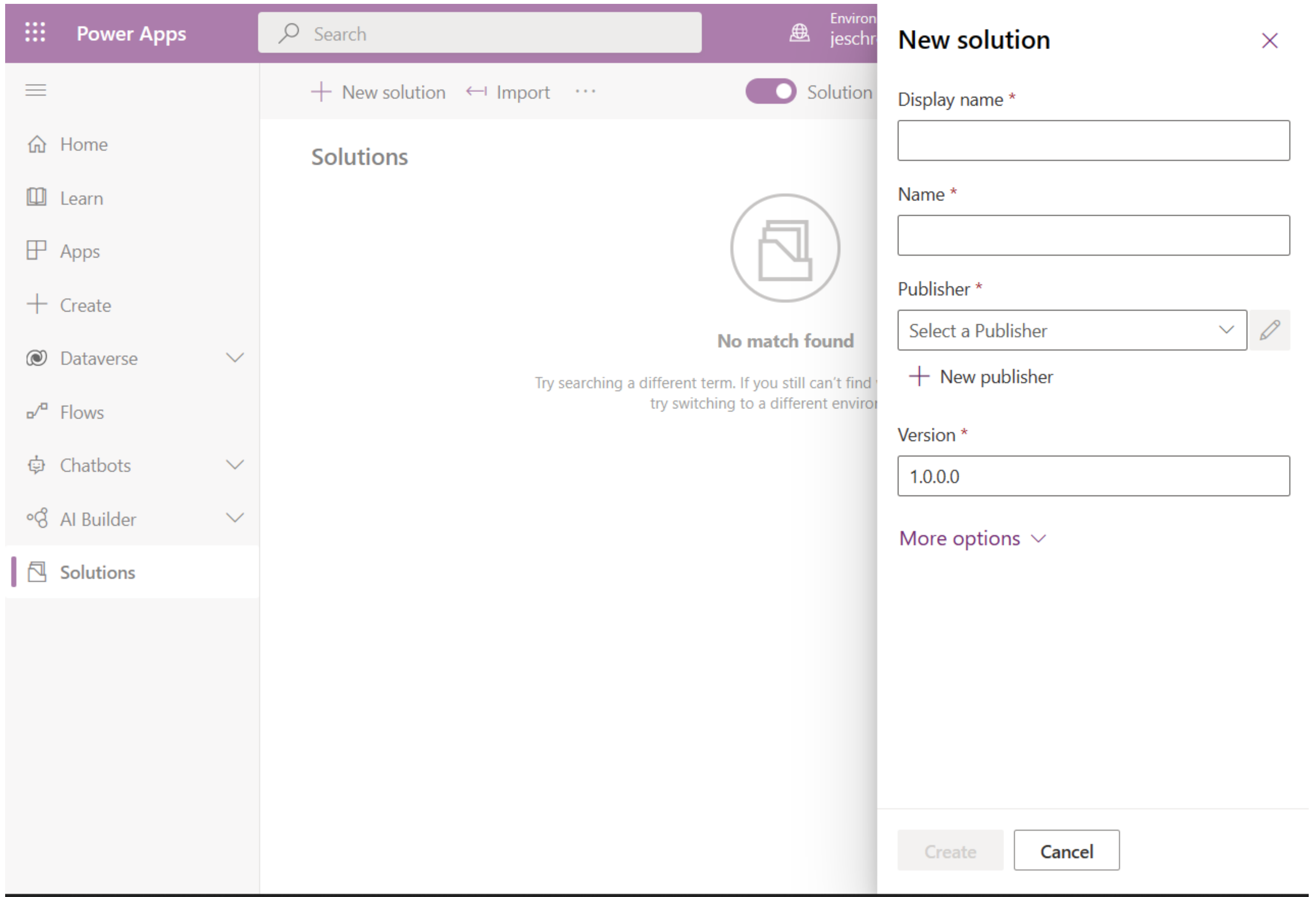
Preview of new object name: new_Object

# 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

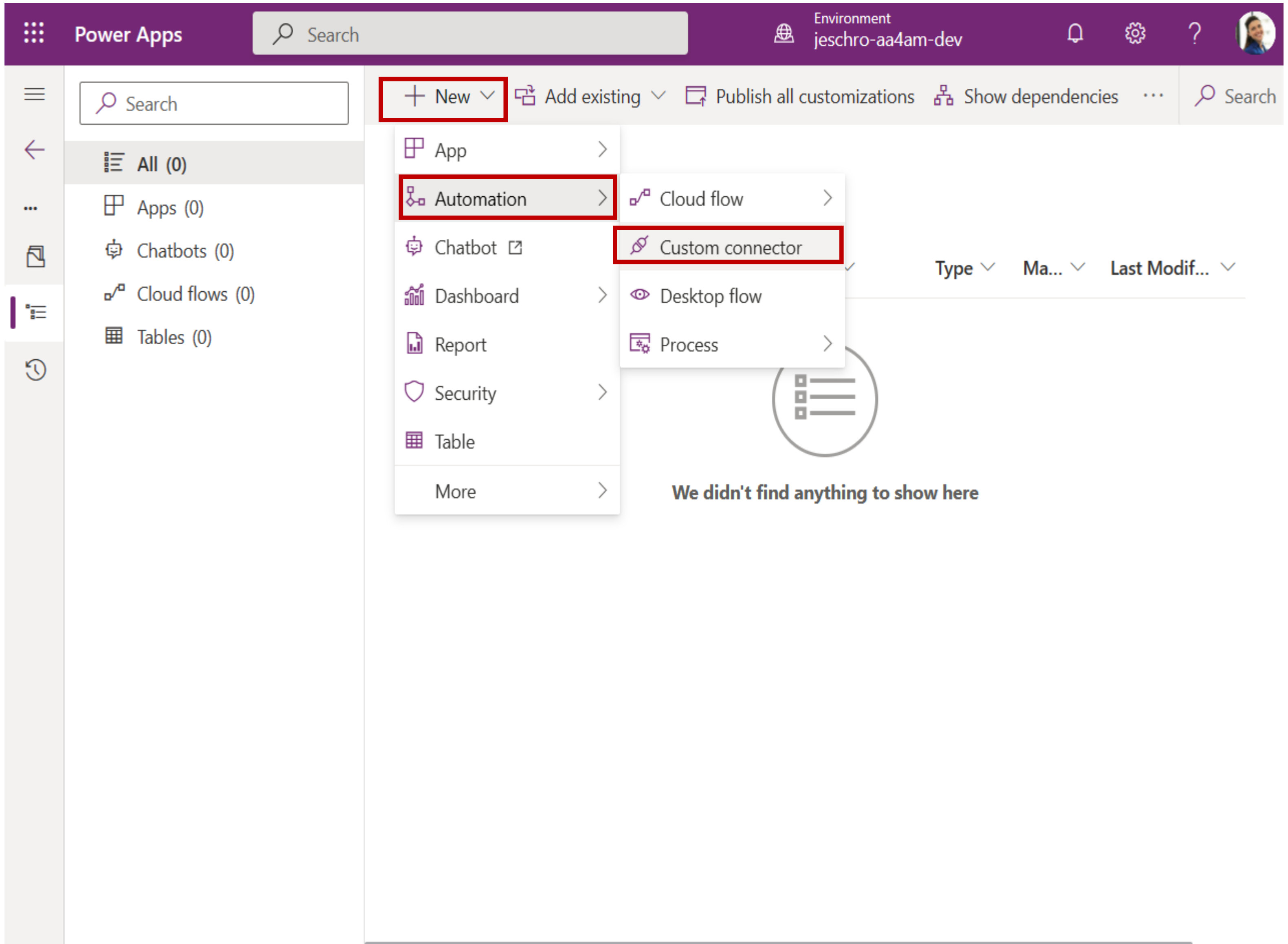
Click Create and click you newly created solution



# Step 17. Add a custom connector to your solution

Open the solution you just created.

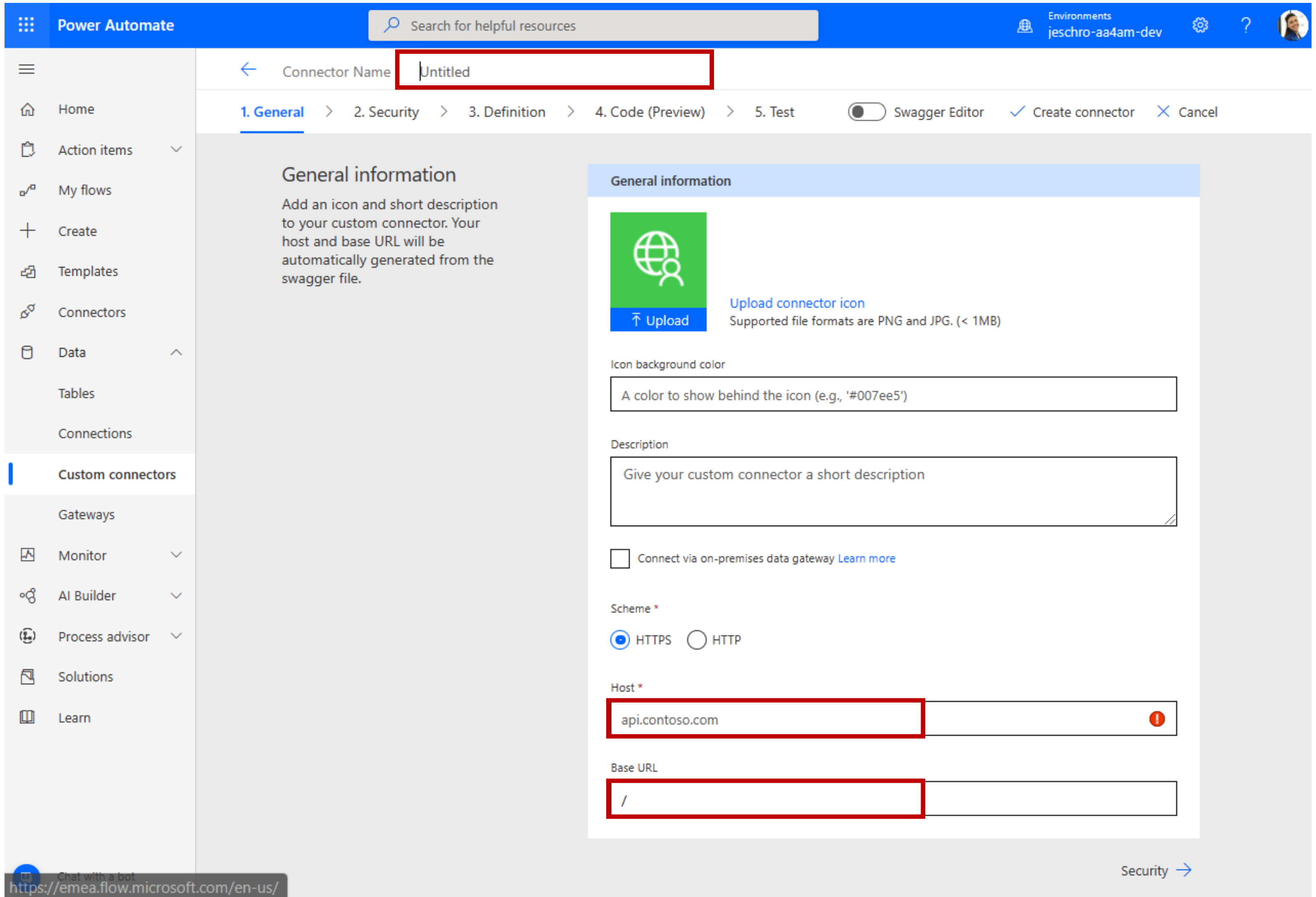
Click New => Automation => Custom connector



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

Enter the following details:

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



Connector Name: Untitled

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

Supported file formats are PNG and JPG. (< 1MB)

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 /

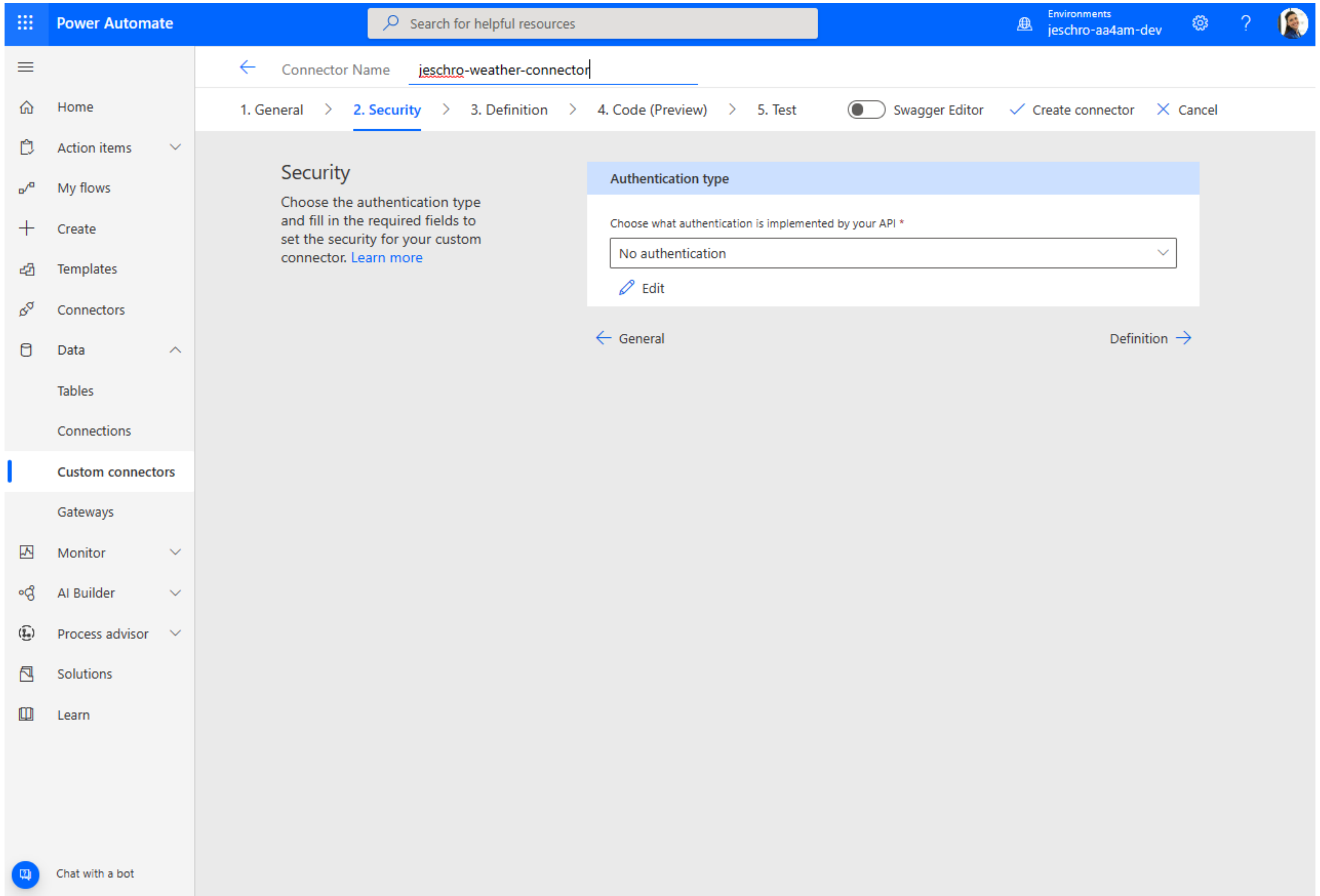
https://emea.flow.microsoft.com/en-us/

## 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.



Connector Name: jeschro-weather-connector

Security

Choose the authentication type and fill in the required fields to set the security for your custom connector. Learn more

Authentication type

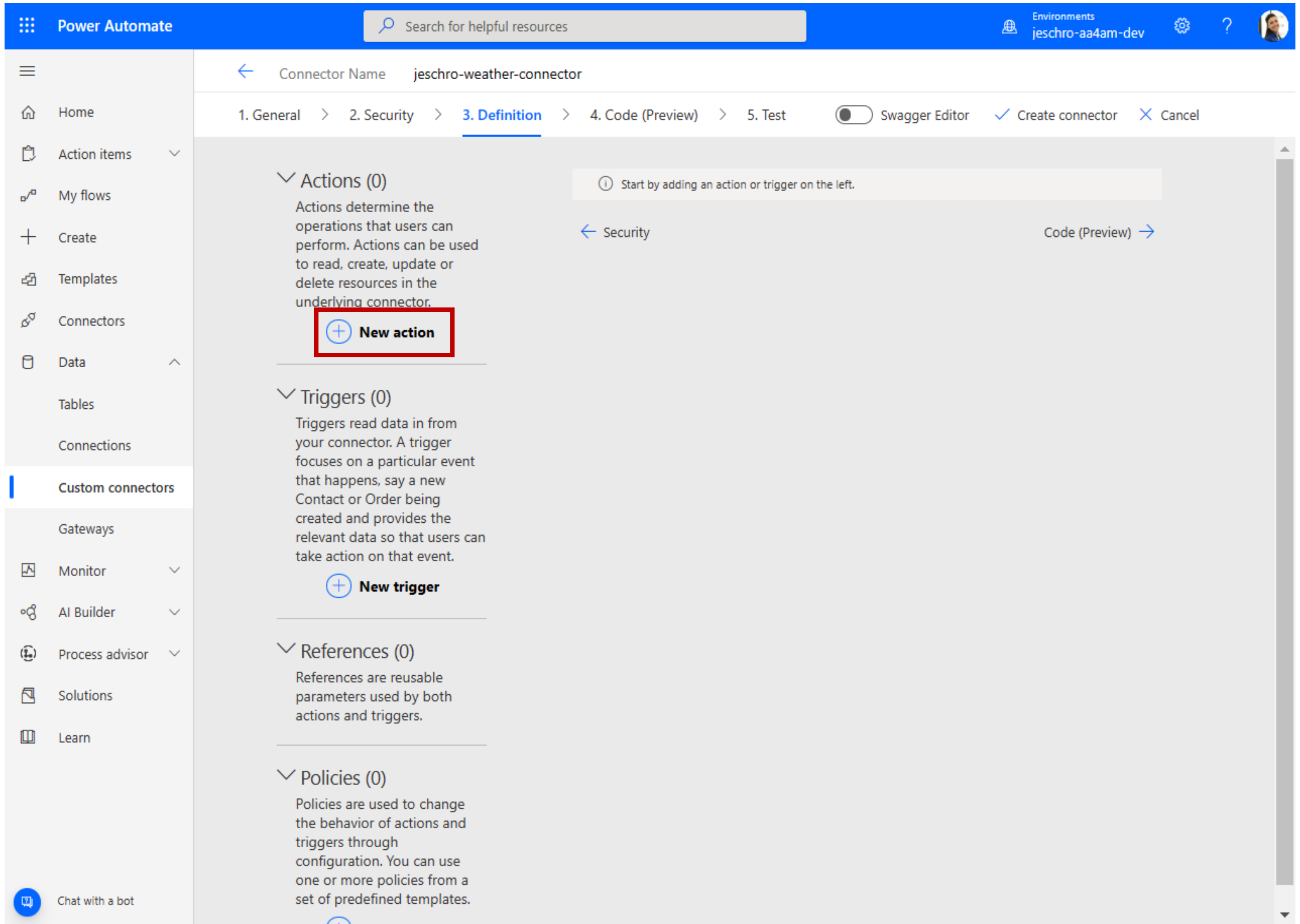
Choose what authentication is implemented by your API *: No authentication

# 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**


Power Automate

Search for helpful resources

Environments
jeschro-aa4am-dev

Home
Action items
My flows
Create
Templates
Connectors
Data
Tables
Connections
Custom connectors
Gateways
Monitor
AI Builder
Process advisor
Solutions
Learn
Chat with a bot

Connector Name jeschro-weather-connector

1. General > 2. Security > 3. Definition > 4. Code (Preview) > 5. Test | Swagger Editor | ✓ Create connector | ✕ Cancel

Actions (0)

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 set of predefined templates.

Start by adding an action or trigger on the left.

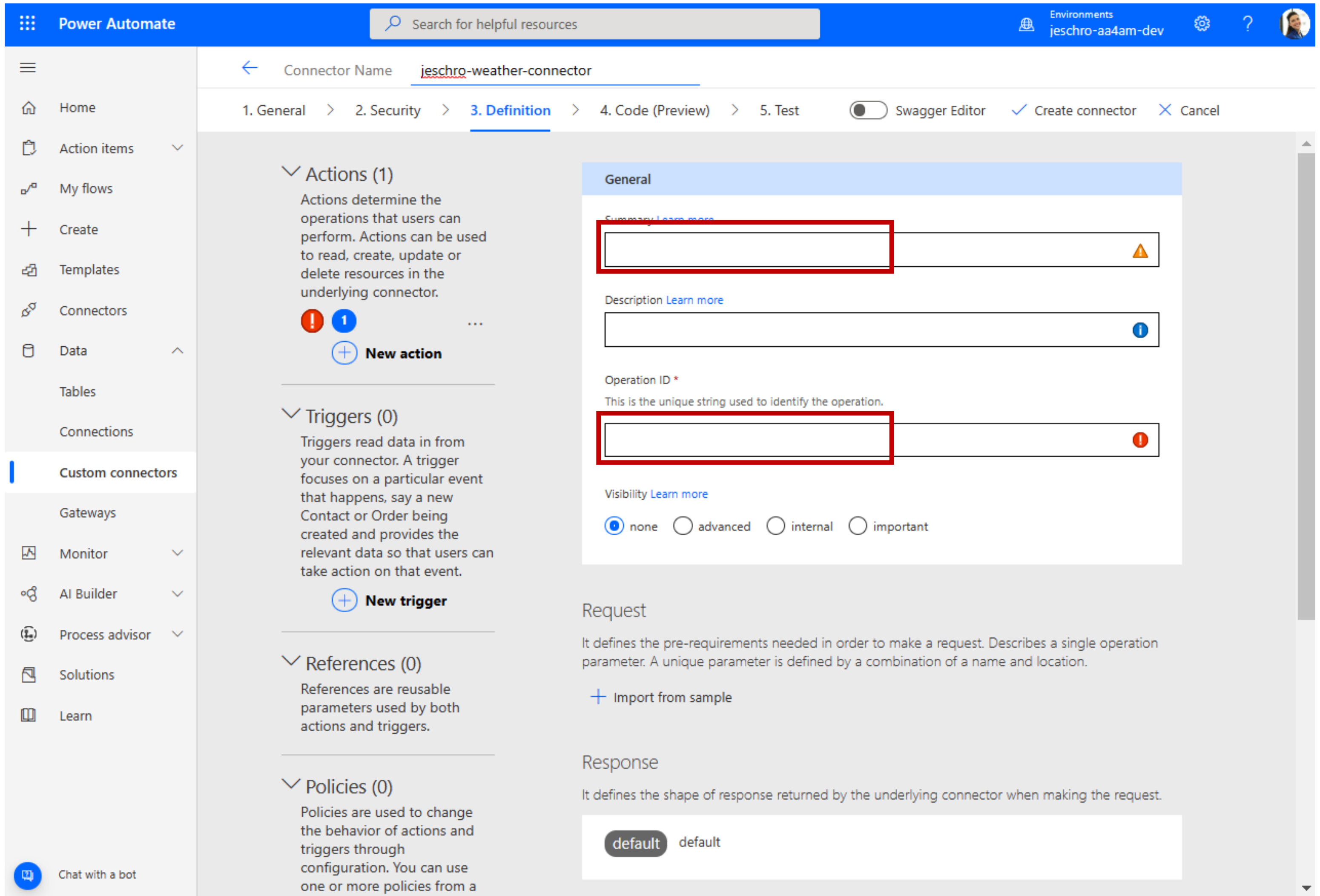
← Security

Code (Preview) →

# 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

Home
Action items
My flows
Create
Templates
Connectors
Data
Tables
Connections
Custom connectors
Gateways
Monitor
AI Builder
Process advisor
Solutions
Learn
Chat with a bot

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.

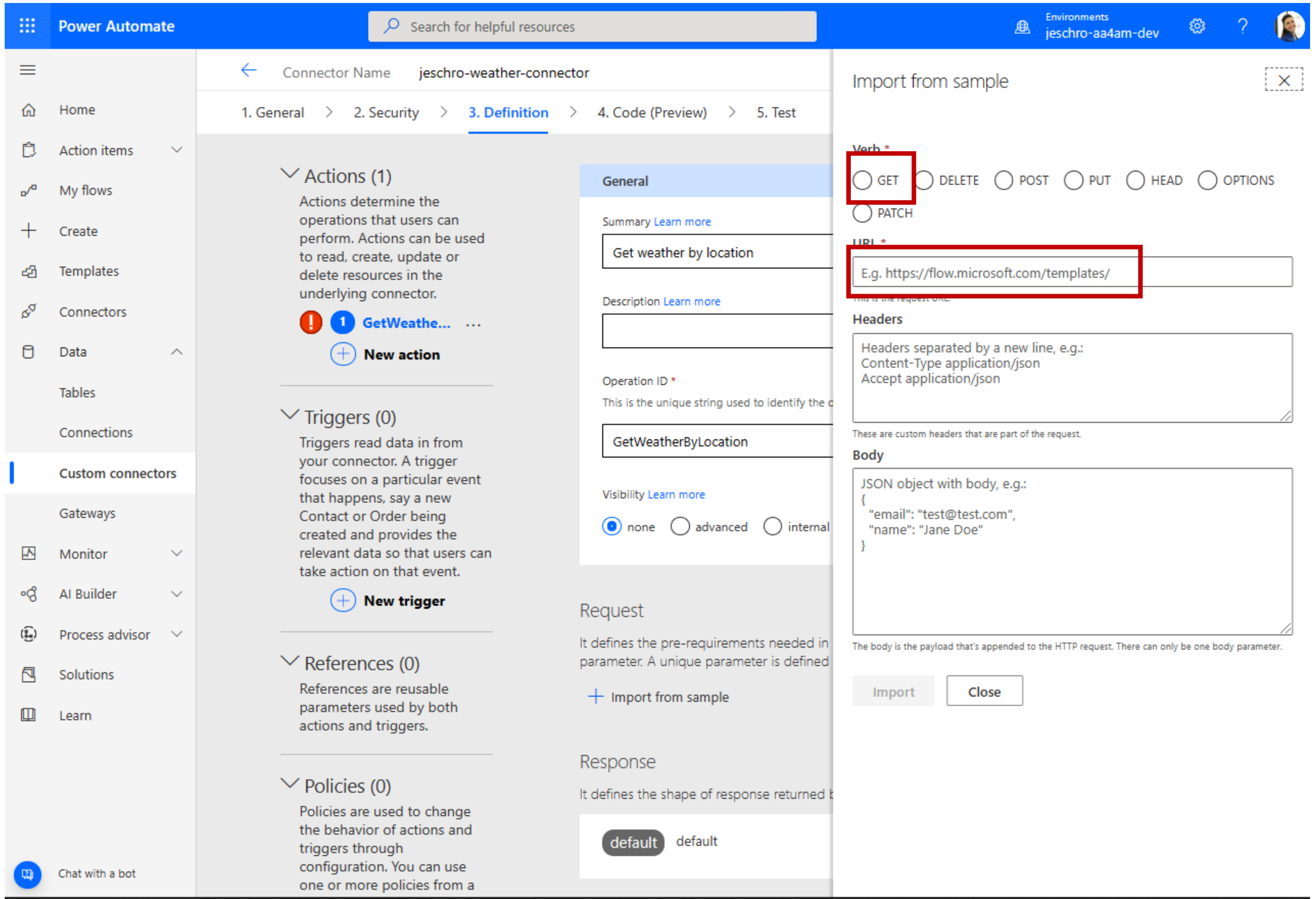
default default

# 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

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.

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.

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: Get weather by location

Operation ID: GetWeatherByLocation

Visibility: [● none] advanced internal

Request

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

Response

It defines the shape of response returned

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

These are custom headers that are part of the request.

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.

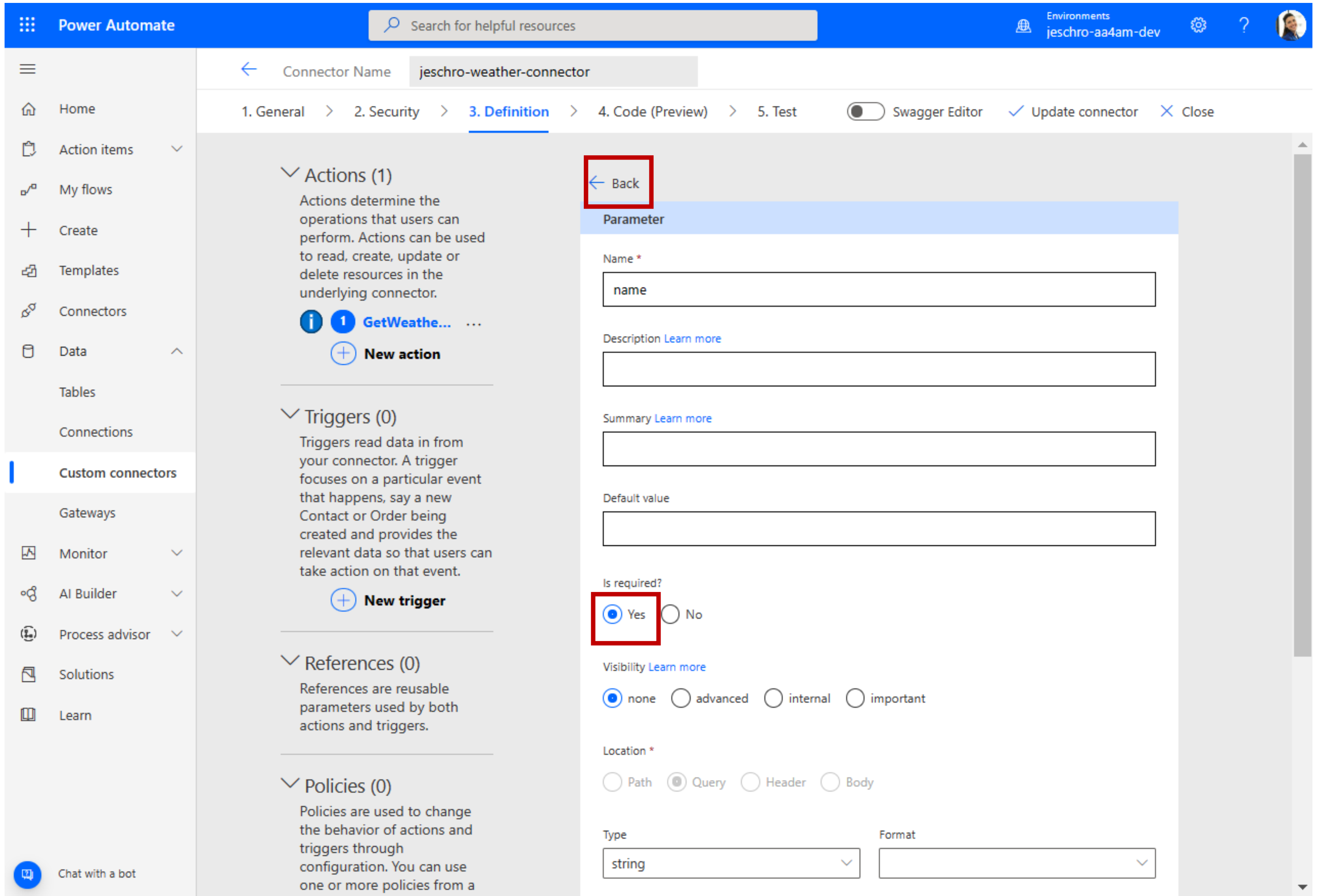
# Step 23. Now define the Request schema of the action - Continued

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



Connector Name: jeschro-weather-connector

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.

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.

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

Name *: name

Description Learn more:

Summary Learn more:

Default value:

Is required?: [✓ Yes] No

Visibility Learn more: [✓ none] advanced internal important

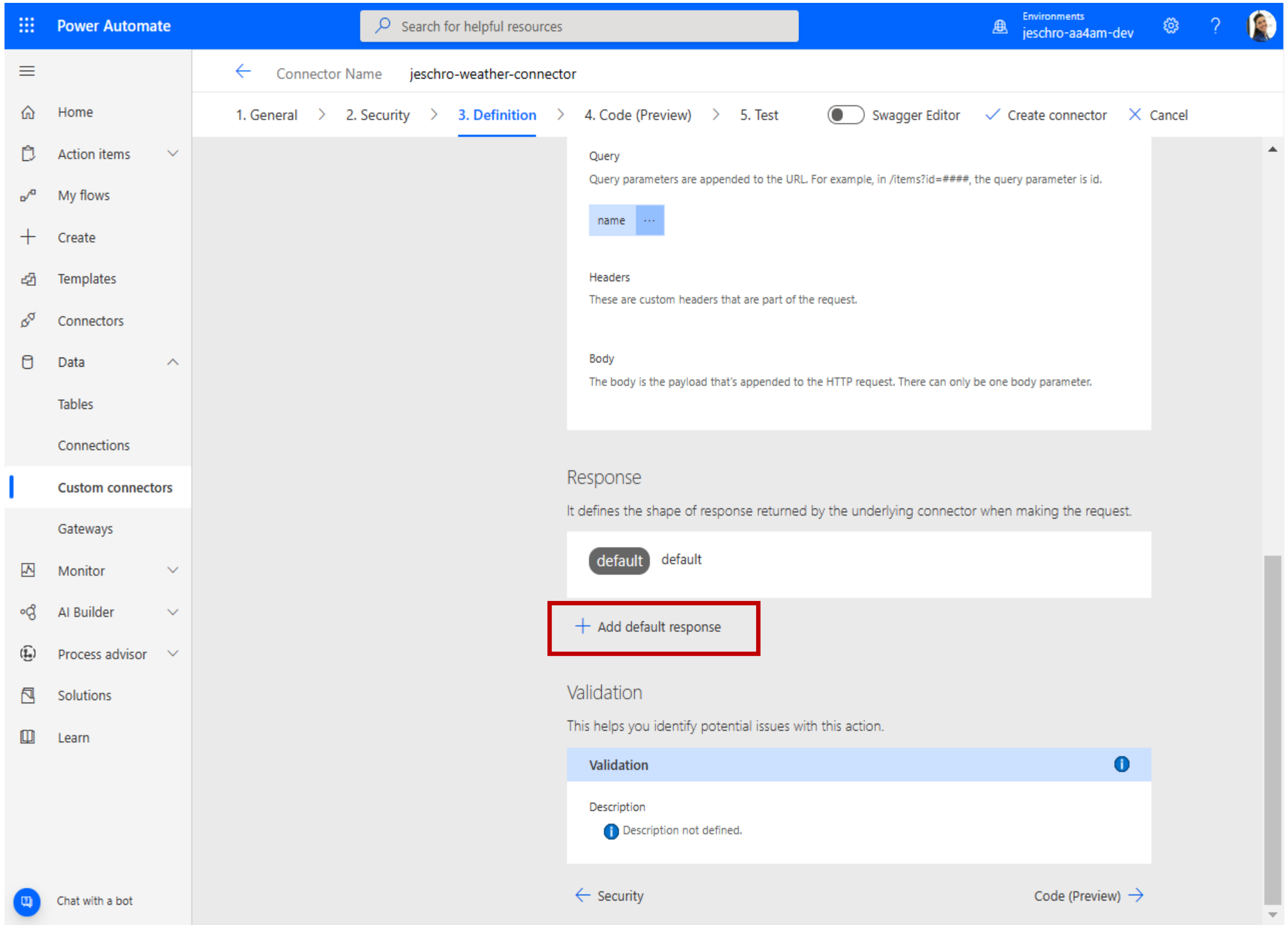
Location *: Path [✓ Query] Header Body

Type: string

Format:

# Step 24. Now define the Response object

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



jeschro-weather-connector

Query

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

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

Validation

This helps you identify potential issues with this action.

Description

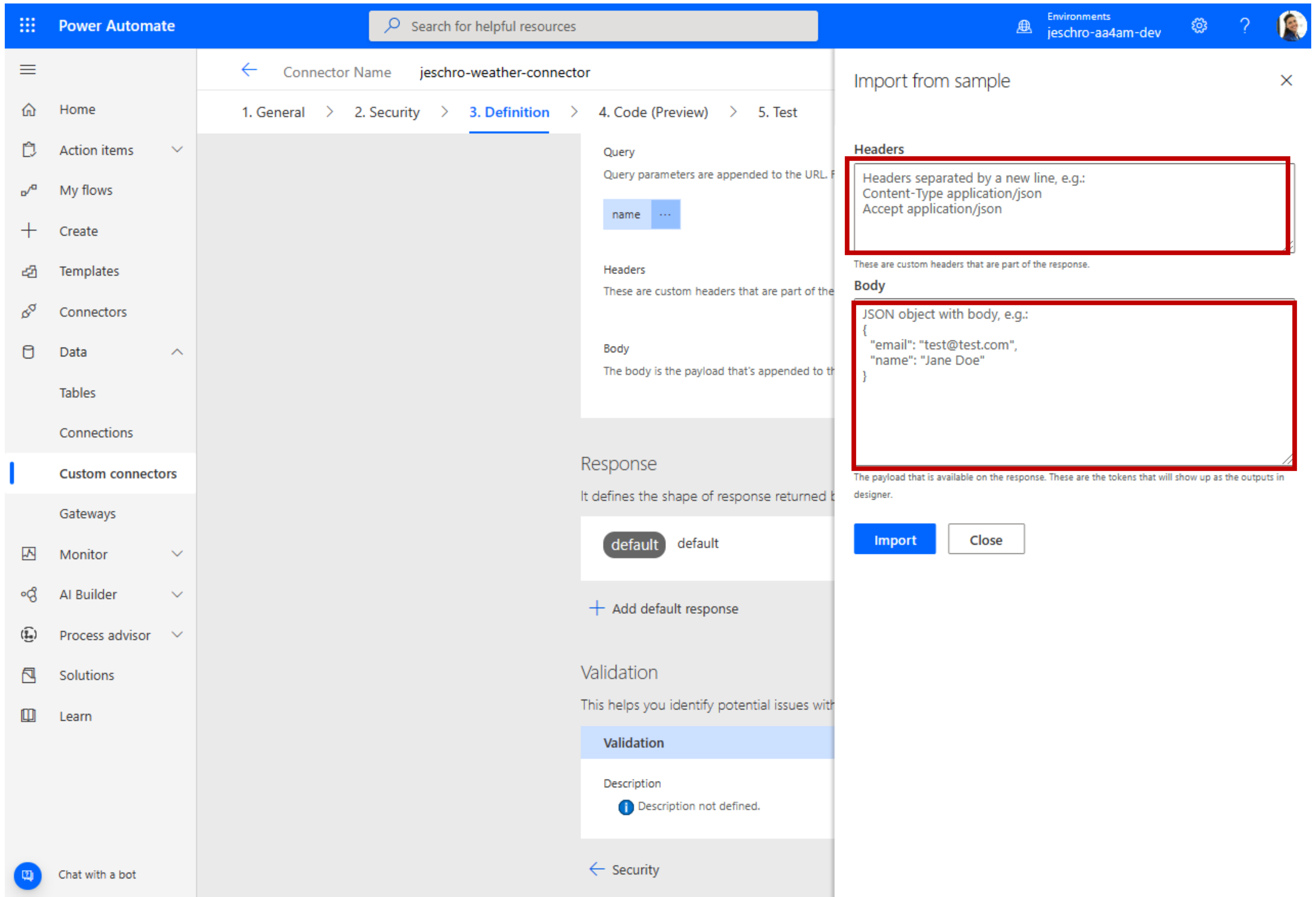
Description not defined.

# 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**



Query parameters are appended to the URL. F

These are custom headers that are part of the

The body is the payload that's appended to th

It defines the shape of response returned b

This helps you identify potential issues wit

Description not defined.

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

These are custom headers that are part of the response.

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

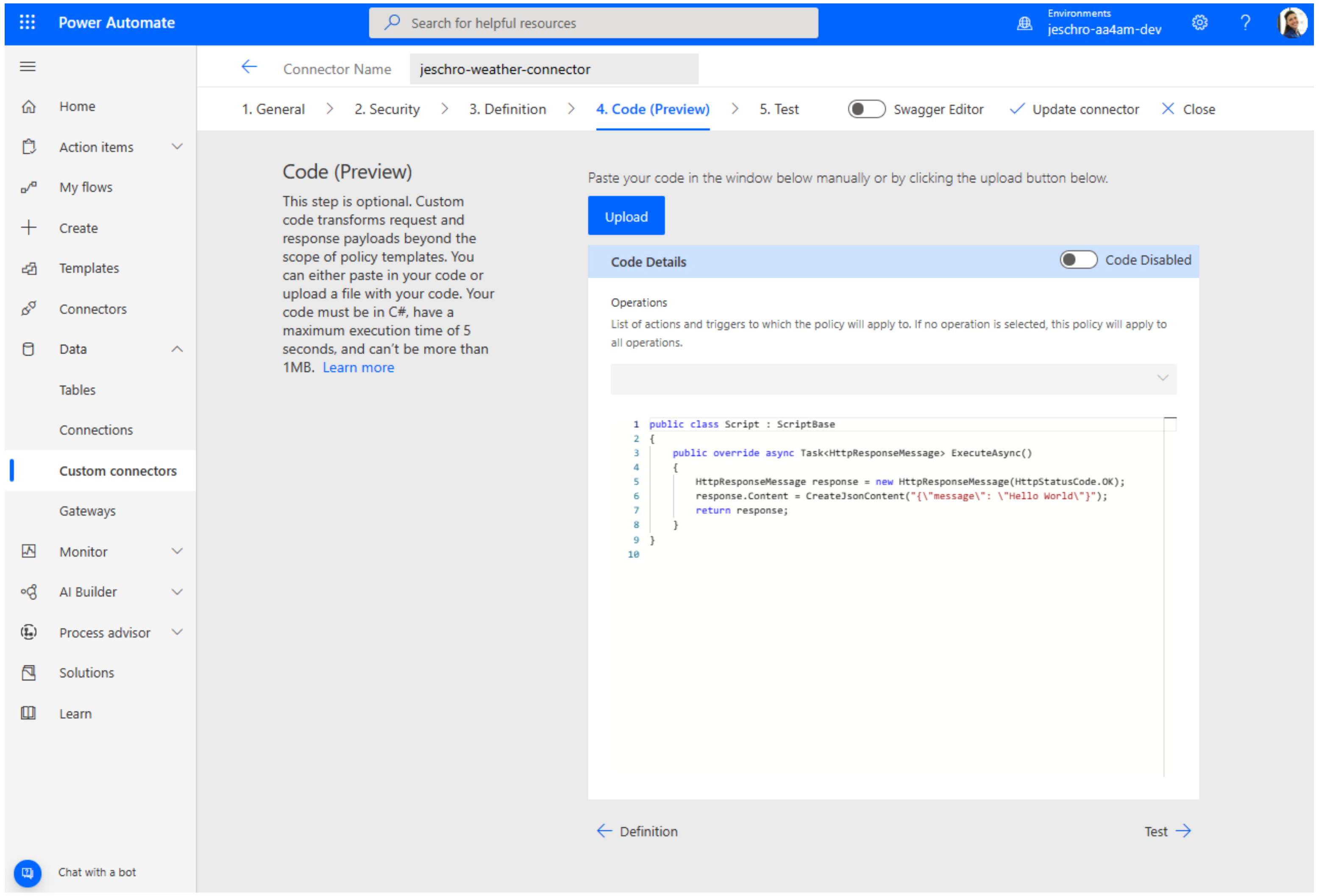
The payload that is available on the response. These are the tokens that will show up as the outputs in designer.

# 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]()



Code (Preview)

This step is optional. Custom code transforms request and response payloads beyond the scope of policy templates. You can either paste in your code or upload a file with your code. Your code must be in C#, have a maximum execution time of 5 seconds, and can't be more than 1MB. Learn more

Paste your code in the window below manually or by clicking the upload button below.

Operations

List of actions and triggers to which the policy will apply to. If no operation is selected, this policy will apply to all operations.

```
public class Script : ScriptBase
{
    public override async Task<HttpResponseMessage> ExecuteAsync()
    {
        HttpResponseMessage response = new HttpResponseMessage(HttpStatusCode.OK);
        response.Content = CreateJsonContent("{\"message\": \"Hello World\"}");
        return response;
    }
}
```

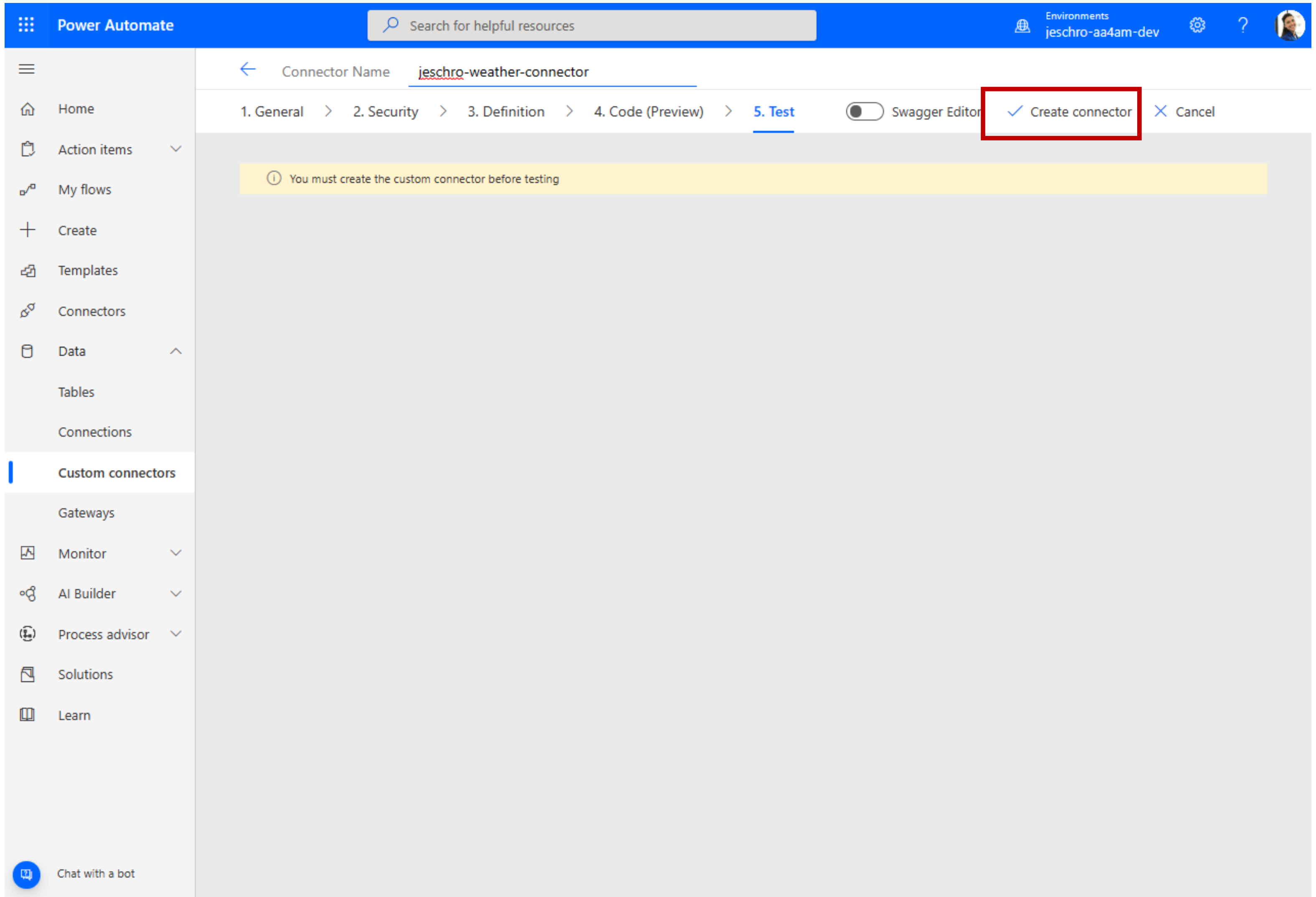
# Step 27. Create/Update your Custom Connector

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



Connector Name jeschro-weather-connector

You must create the custom connector before testing

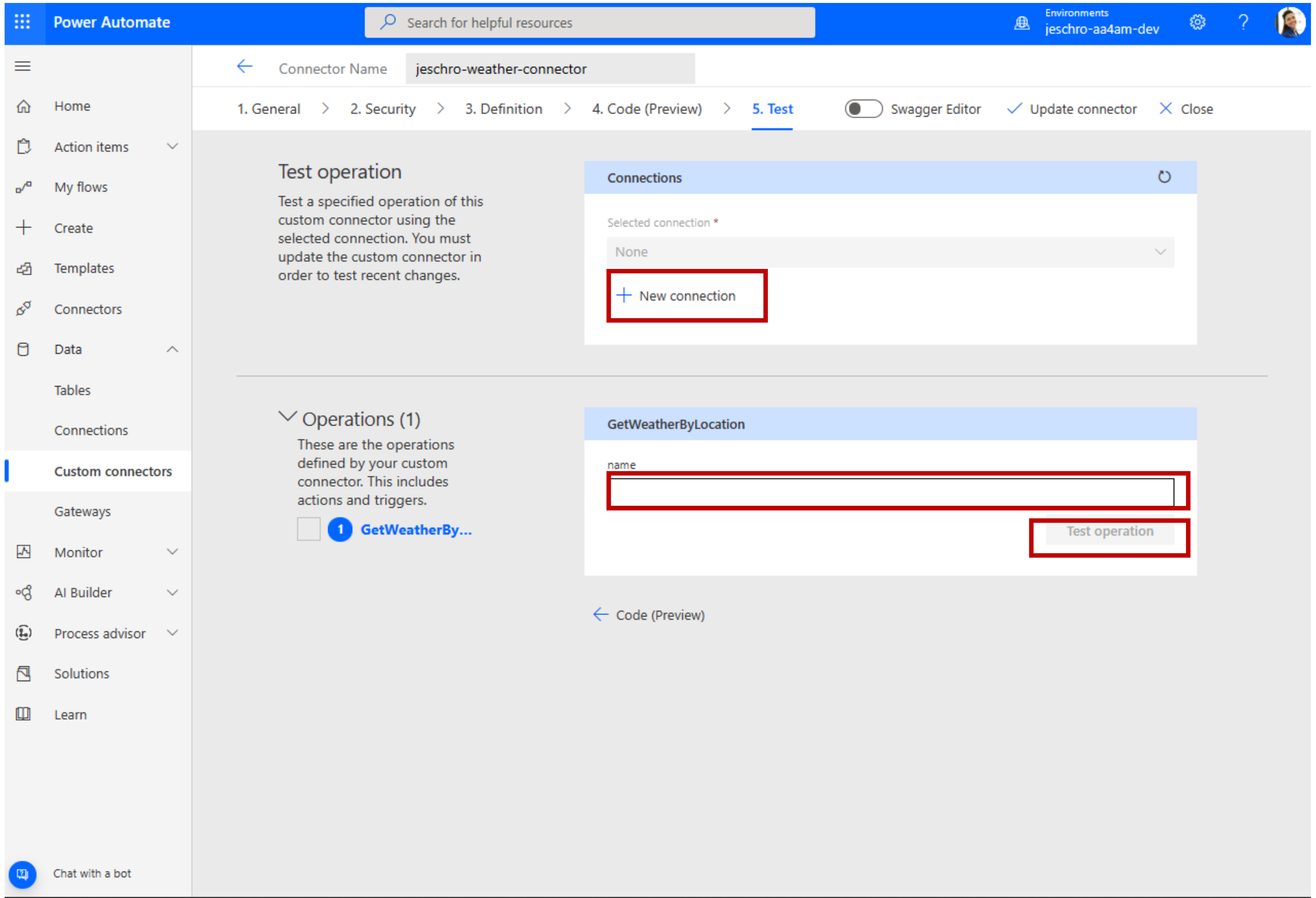
# Step 28. Test your Custom Connector

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**



Connector Name: jeschro-weather-connector

Test operation

Test a specified operation of this custom connector using the selected connection. You must update the custom connector in order to test recent changes.

Selected connection *: None

Operations (1)

These are the operations defined by your custom connector. This includes actions and triggers.

GetWeatherByLocation

name

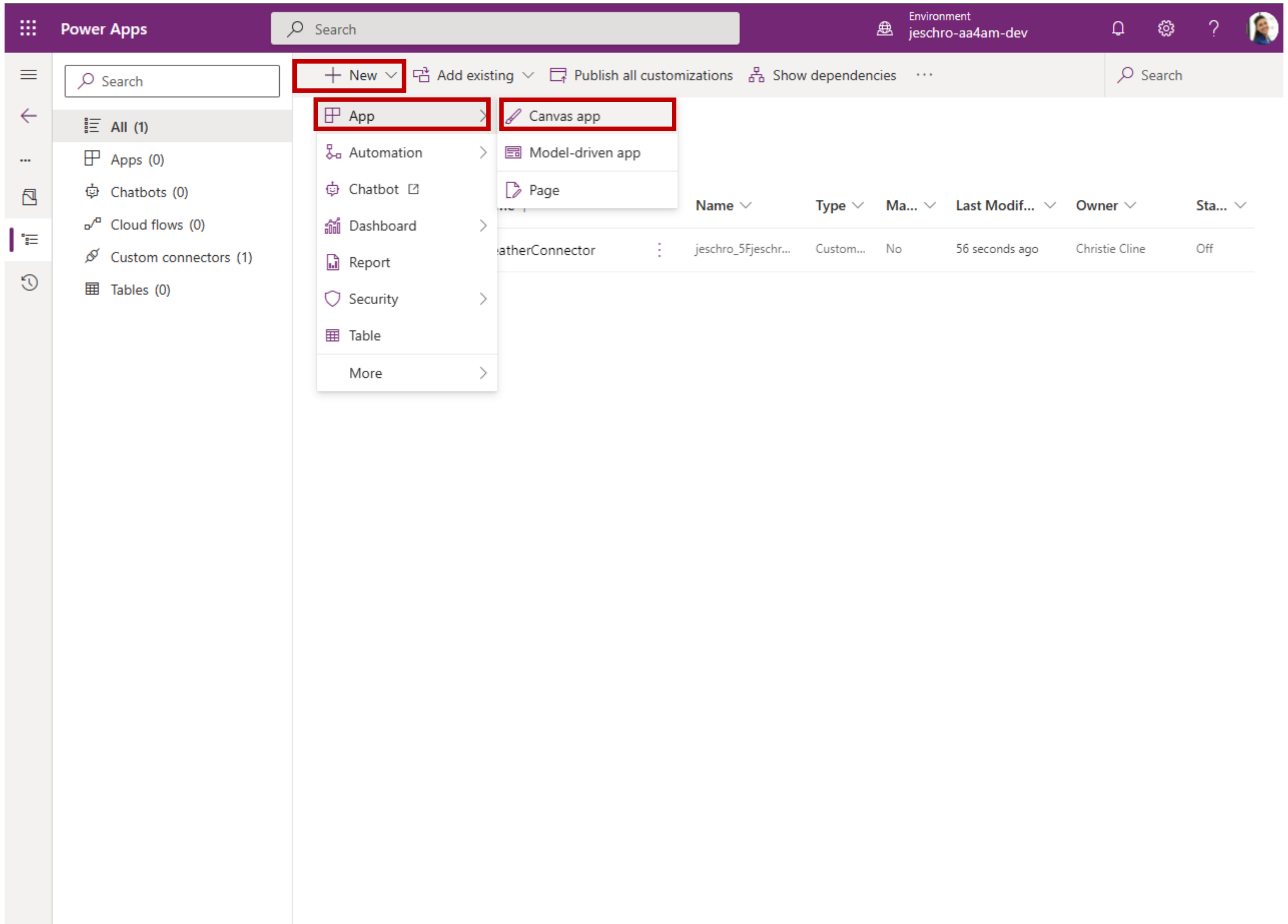
## Using a Power Platform custom connector in a Power App

# 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**



| Name | Type | Ma... | Last Modif... | Owner | Sta... |
|---|---|---|---|---|---|
| atherConnector jeschro_5Fjeschr... | Custom... | No | 56 seconds ago | Christie Cline | Off |

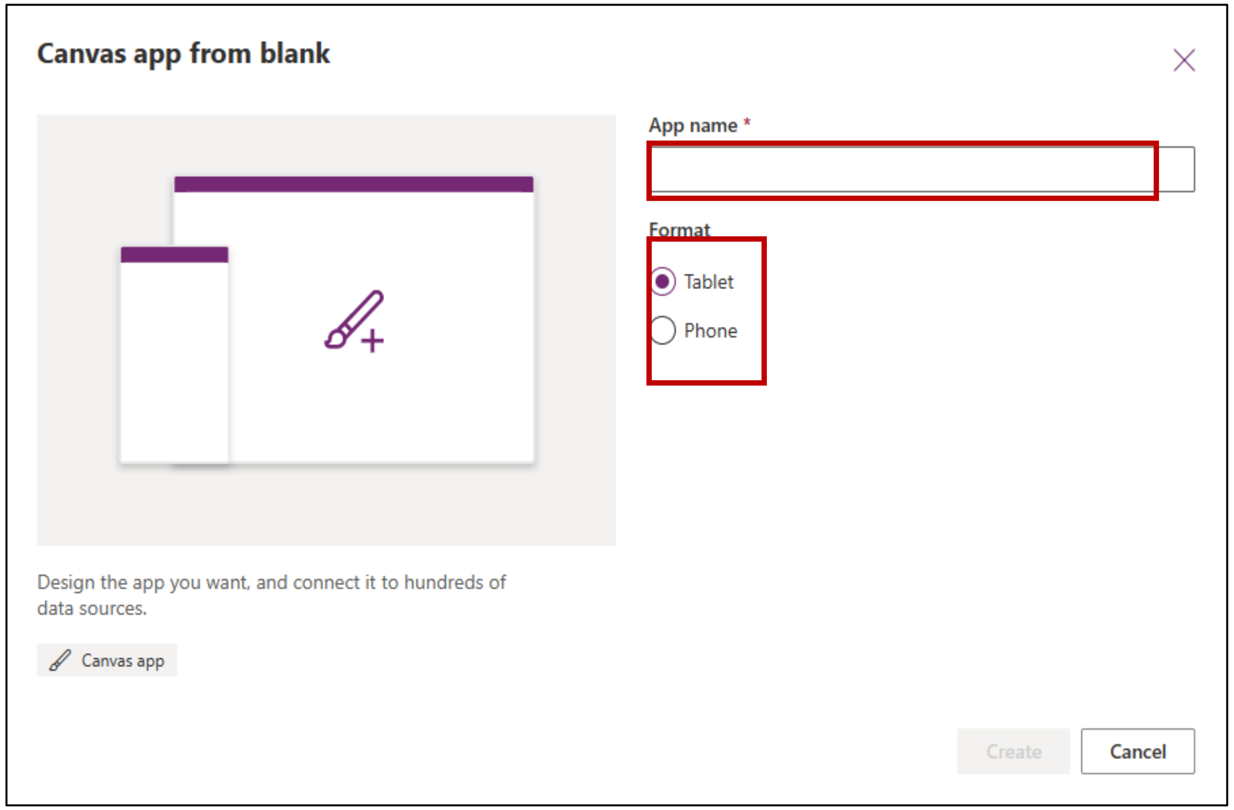
## Step 30. Name you app and select format

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 *

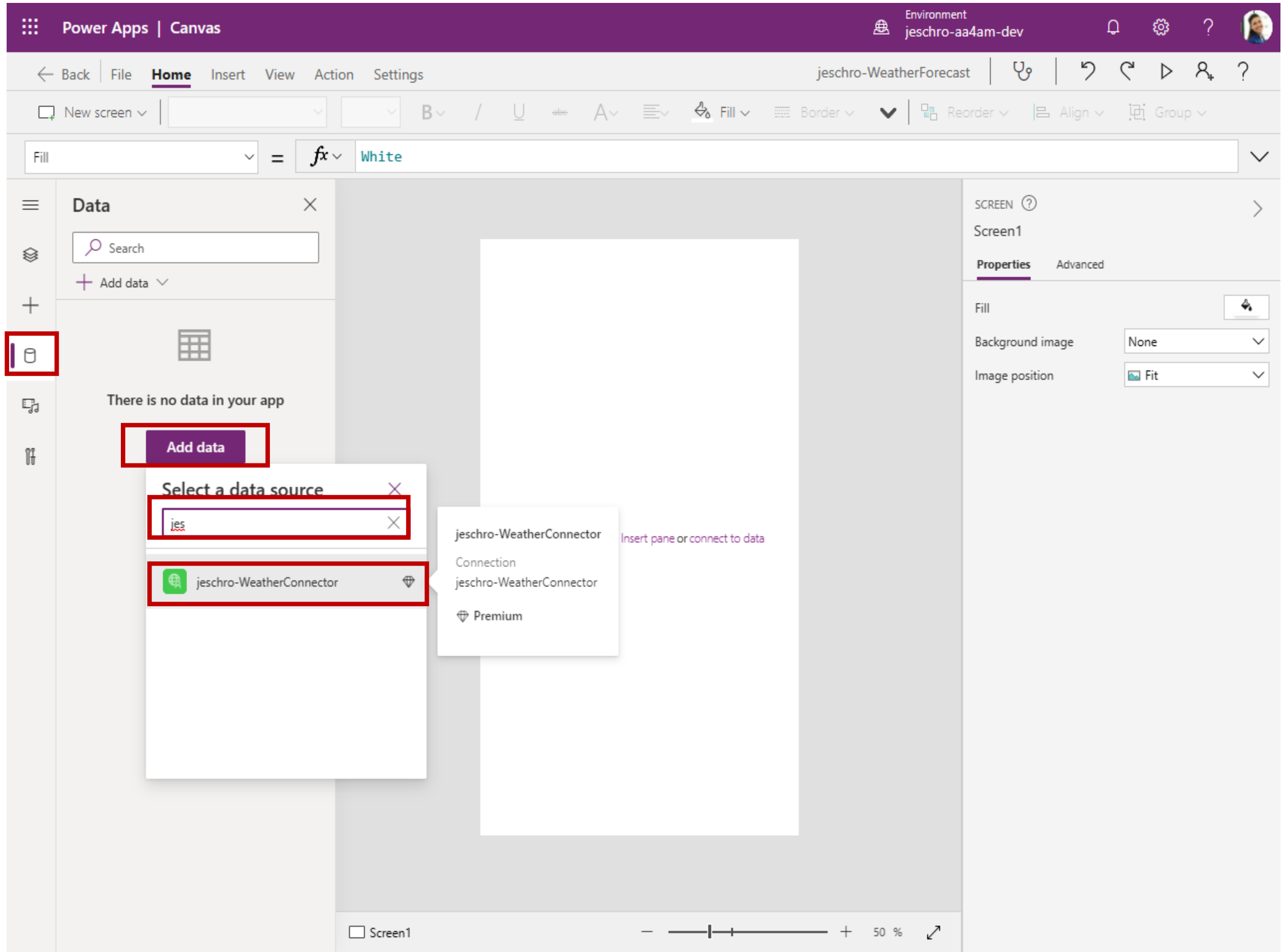
Format

(● Tablet) ○ Phone

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

# Step 31. Add your custom connector as a datasource

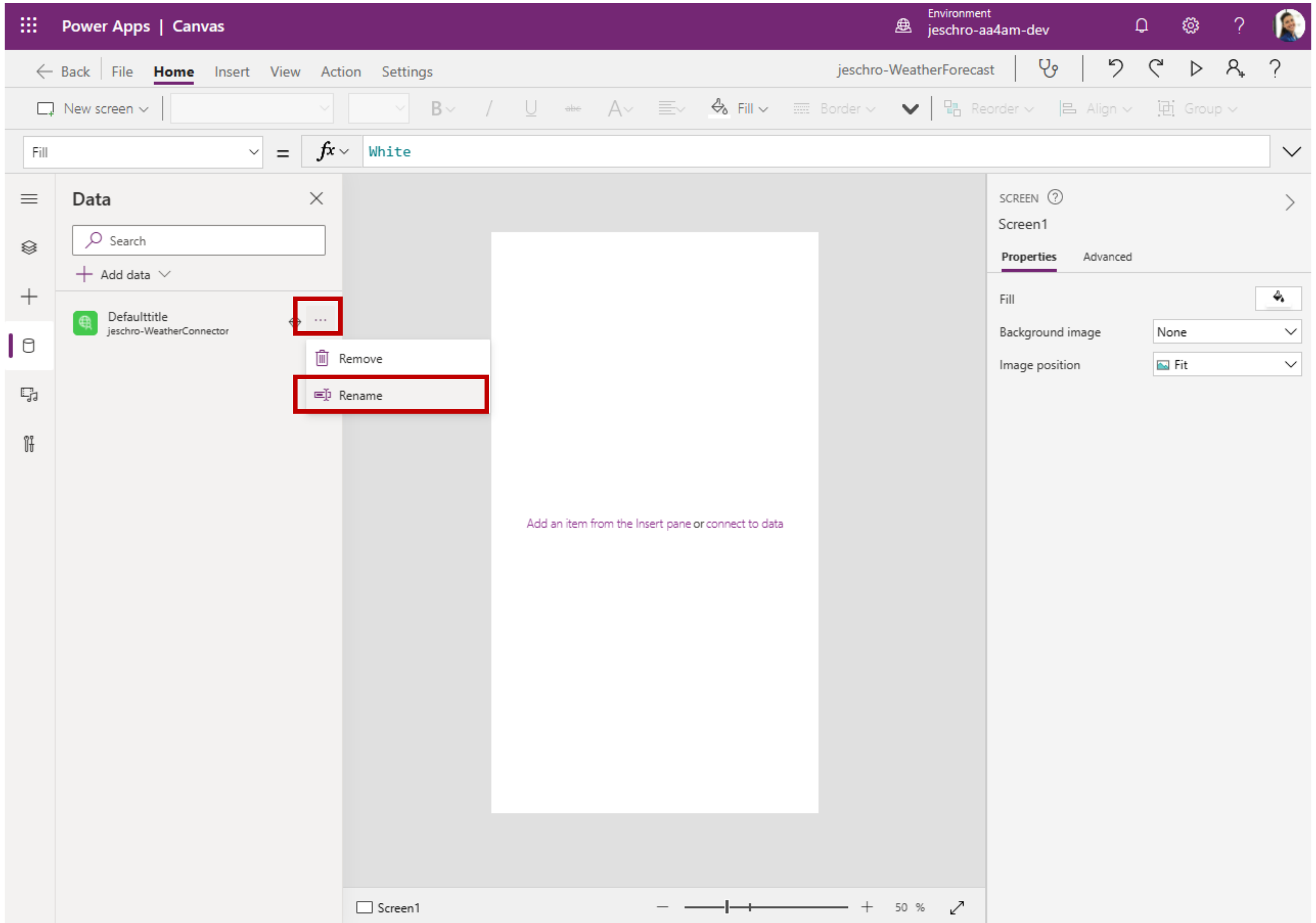
1. Click the database icon on the left navigation bar
2. Click **Add data**
3. Search for your alias to find your connector
4. Click your connector and click it again



# Step 32. Rename the reference to your connector

Your connector is added with Defaulttitle as name.

Change this to WeatherConnector

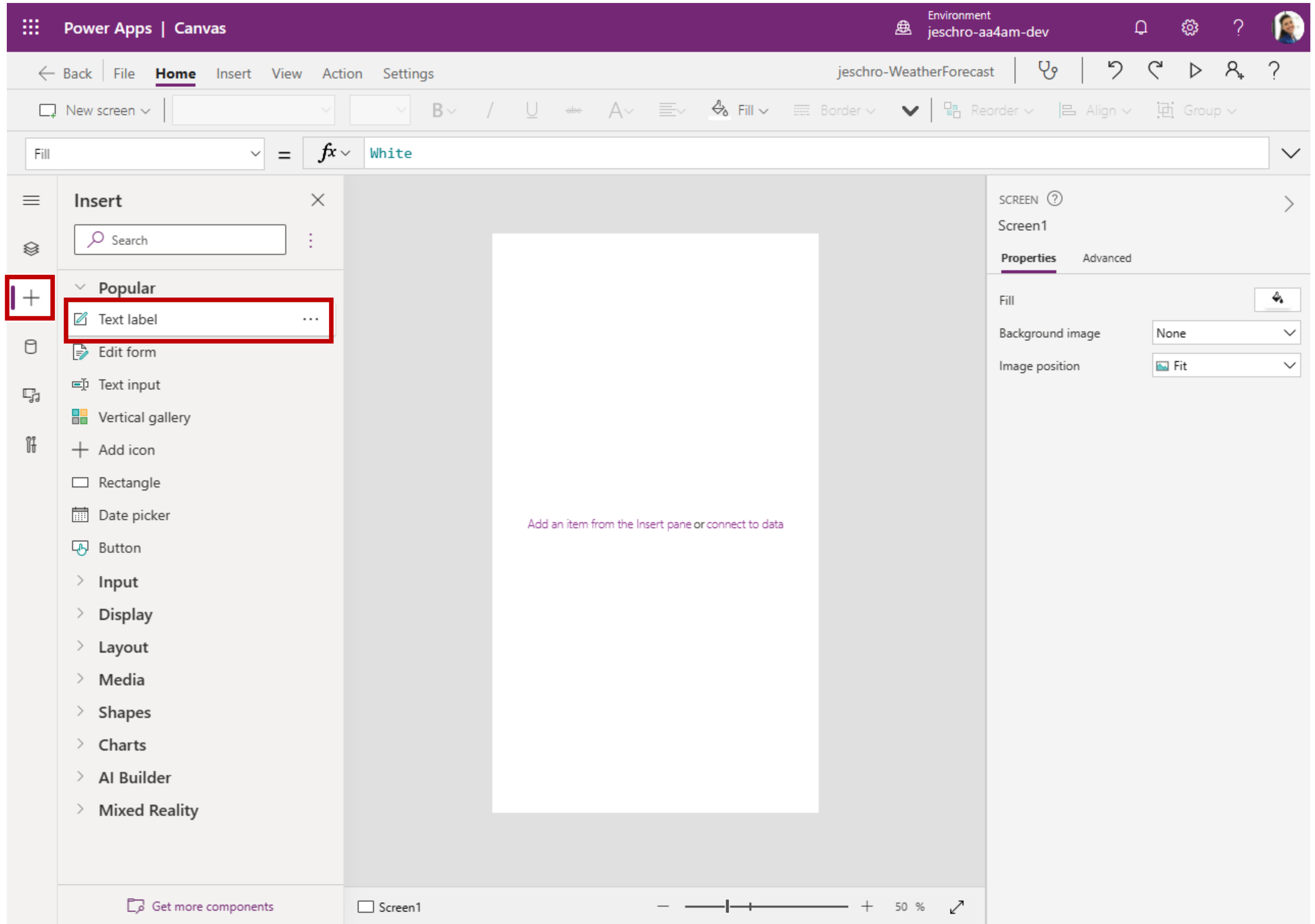


# Step 33. Add controls to your app

Add a label to your app

Click the **+** icon in the left Navigation bar

Click Text label



Fill = White

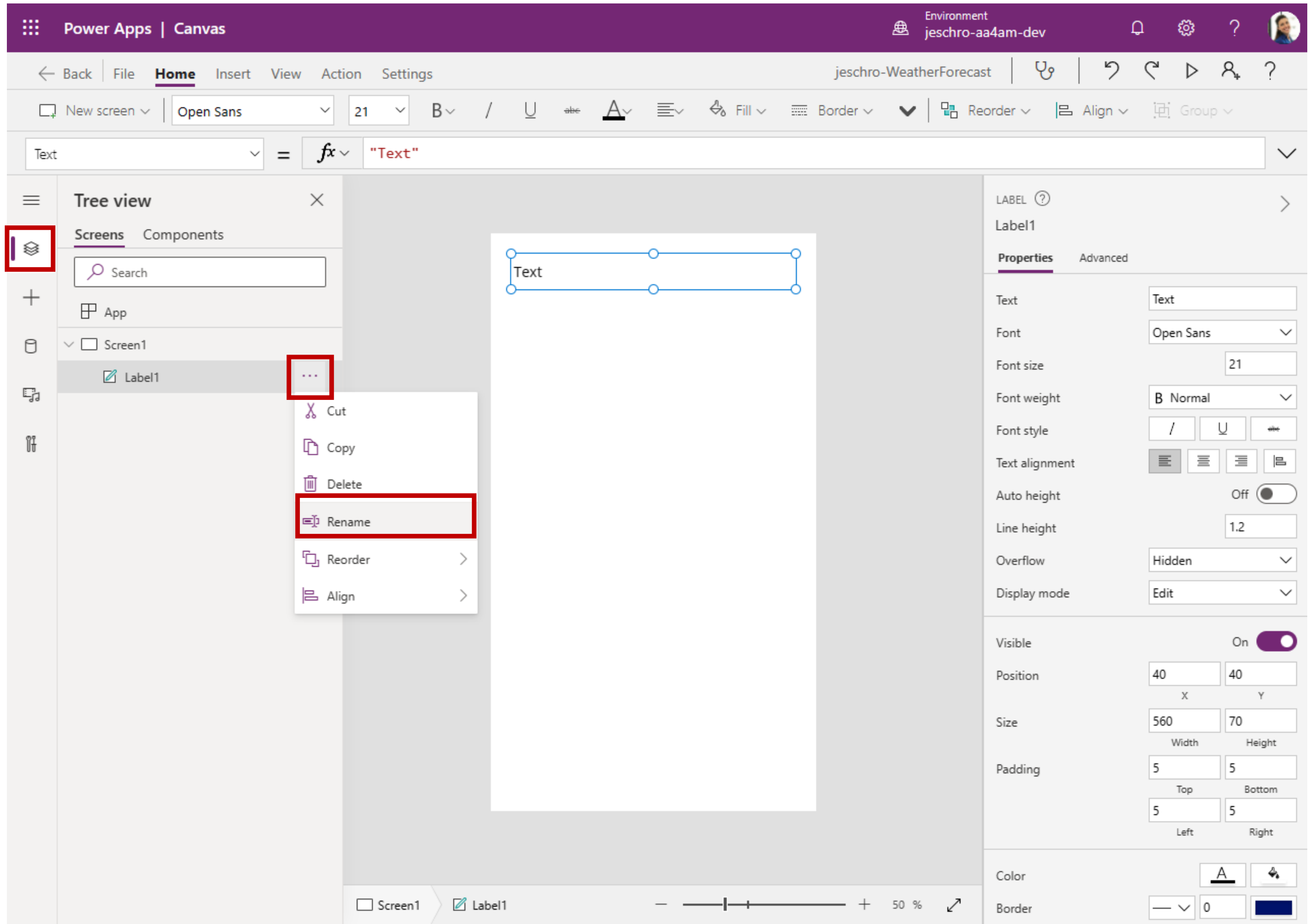
# Step 34. Set the label control properties

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



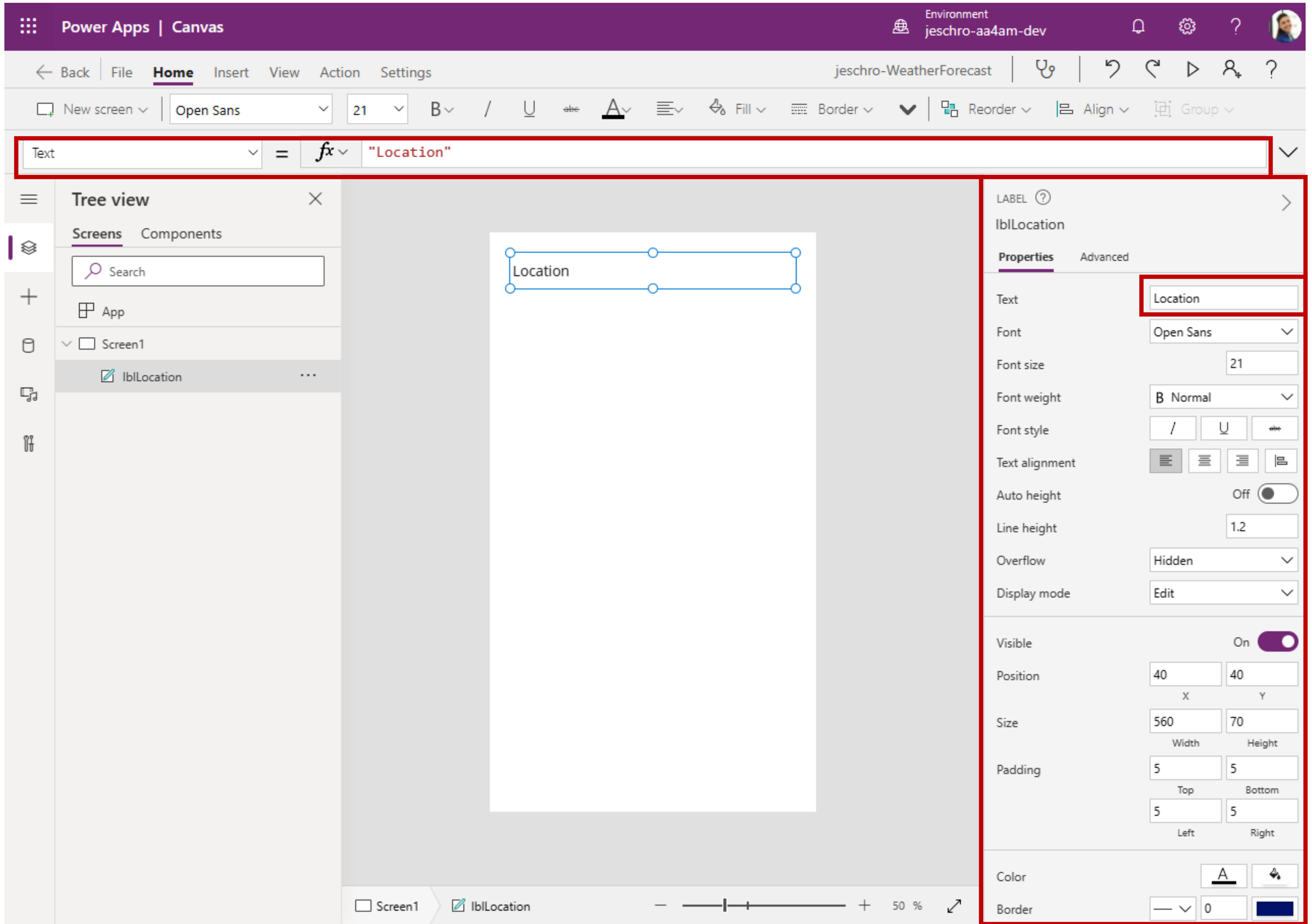
| | |
|---|---|
| Text | Text |
| Font | Open Sans |
| Font size | 21 |
| Font weight | B Normal |
| Font style | |
| Text alignment | |
| Auto height | Off |
| Line height | 1.2 |
| Overflow | Hidden |
| Display mode | Edit |
| Visible | On |
| Position | 40 (X), 40 (Y) |
| Size | 560 (Width), 70 (Height) |
| Padding | 5 (Top), 5 (Bottom), 5 (Left), 5 (Right) |
| Color | |
| Border | 0 |

# 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**



Text = "Location"

LABEL
lblLocation
Properties | Advanced

| | |
|---|---|
| Text | Location |
| Font | Open Sans |
| Font size | 21 |
| Font weight | B Normal |
| Font style | / U |
| Text alignment | |
| Auto height | Off |
| Line height | 1.2 |
| Overflow | Hidden |
| Display mode | Edit |
| Visible | On |
| Position | 40 (X), 40 (Y) |
| Size | 560 (Width), 70 (Height) |
| Padding | 5 (Top), 5 (Bottom), 5 (Left), 5 (Right) |
| Color | |
| Border | 0 |

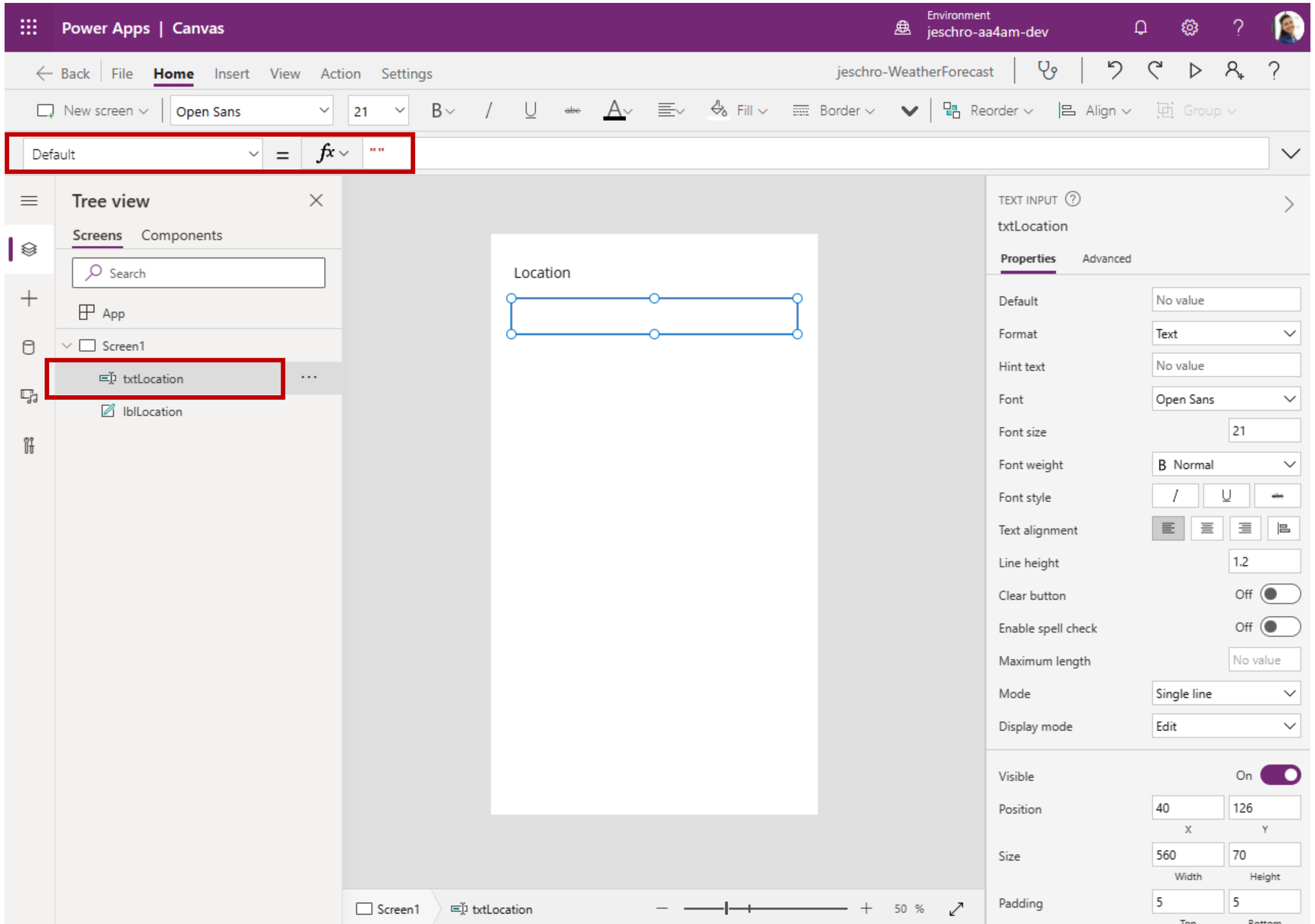
# Step 36. Add Text input control to your app

Set the following properties for the Text Input control

Name: txtLocation

Default: ""

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



TEXT INPUT
txtLocation

Properties | Advanced

| | |
|---|---|
| Default | No value |
| Format | Text |
| Hint text | No value |
| Font | Open Sans |
| Font size | 21 |
| Font weight | B Normal |
| Font style | |
| Text alignment | |
| Line height | 1.2 |
| Clear button | Off |
| Enable spell check | Off |
| Maximum length | No value |
| Mode | Single line |
| Display mode | Edit |
| Visible | On |
| Position | 40 (X), 126 (Y) |
| Size | 560 (Width), 70 (Height) |
| Padding | 5 (Top), 5 (Bottom) |

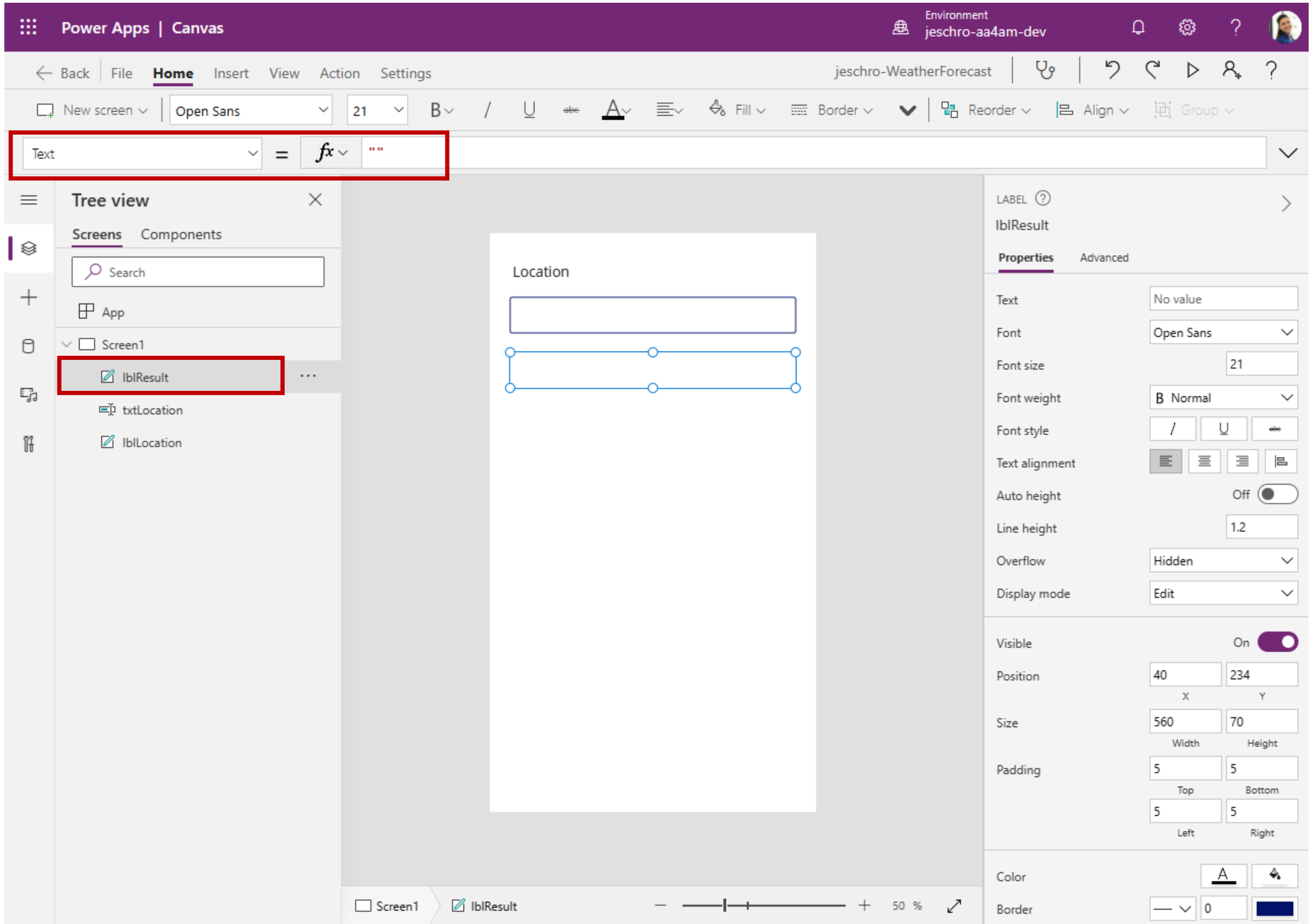
# Step 37. Add label to display the forecast result

Set the following properties for the label control

Name: lblResult

Text: ""

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



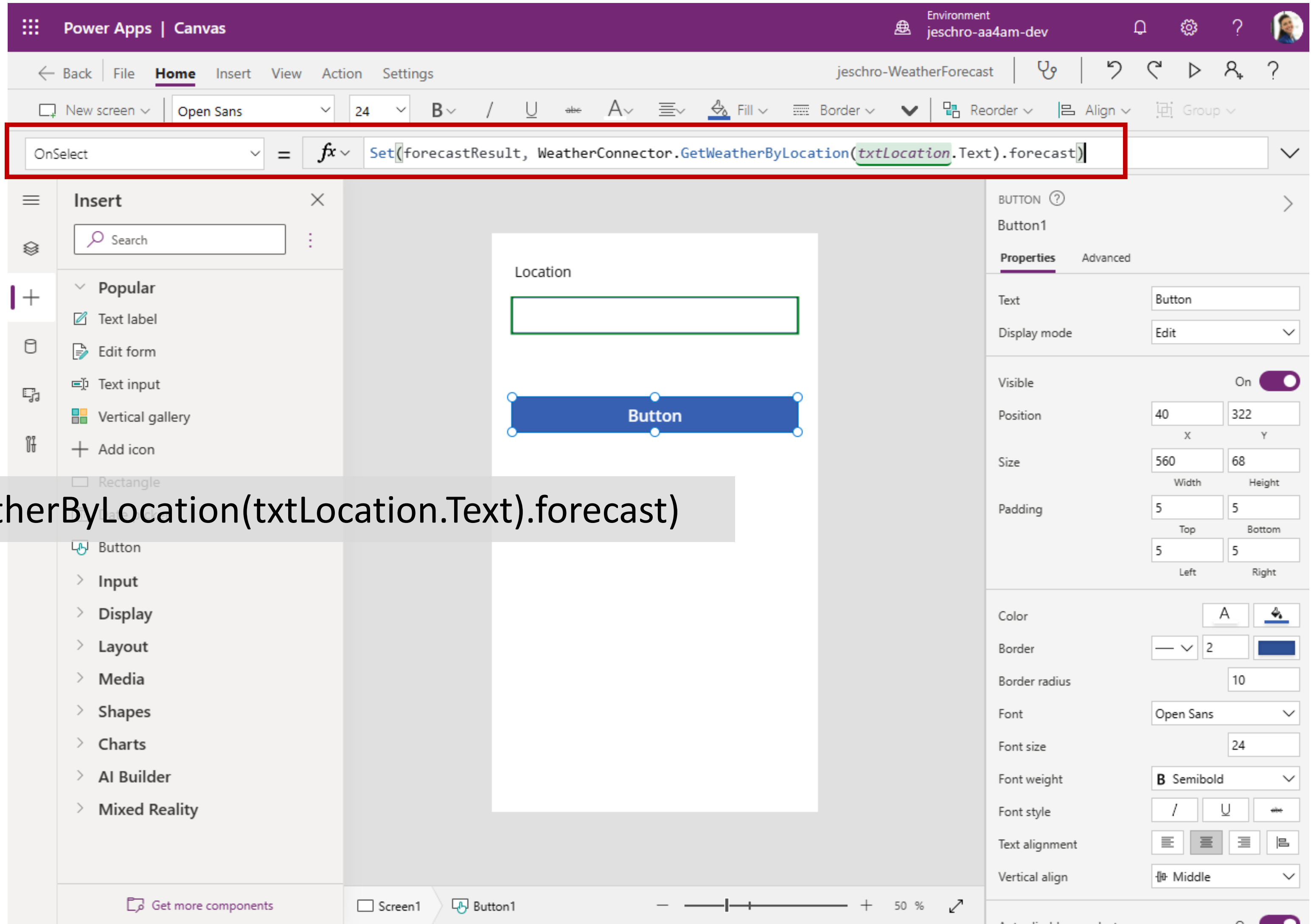
## Step 38. Add a button to call our custom connector

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



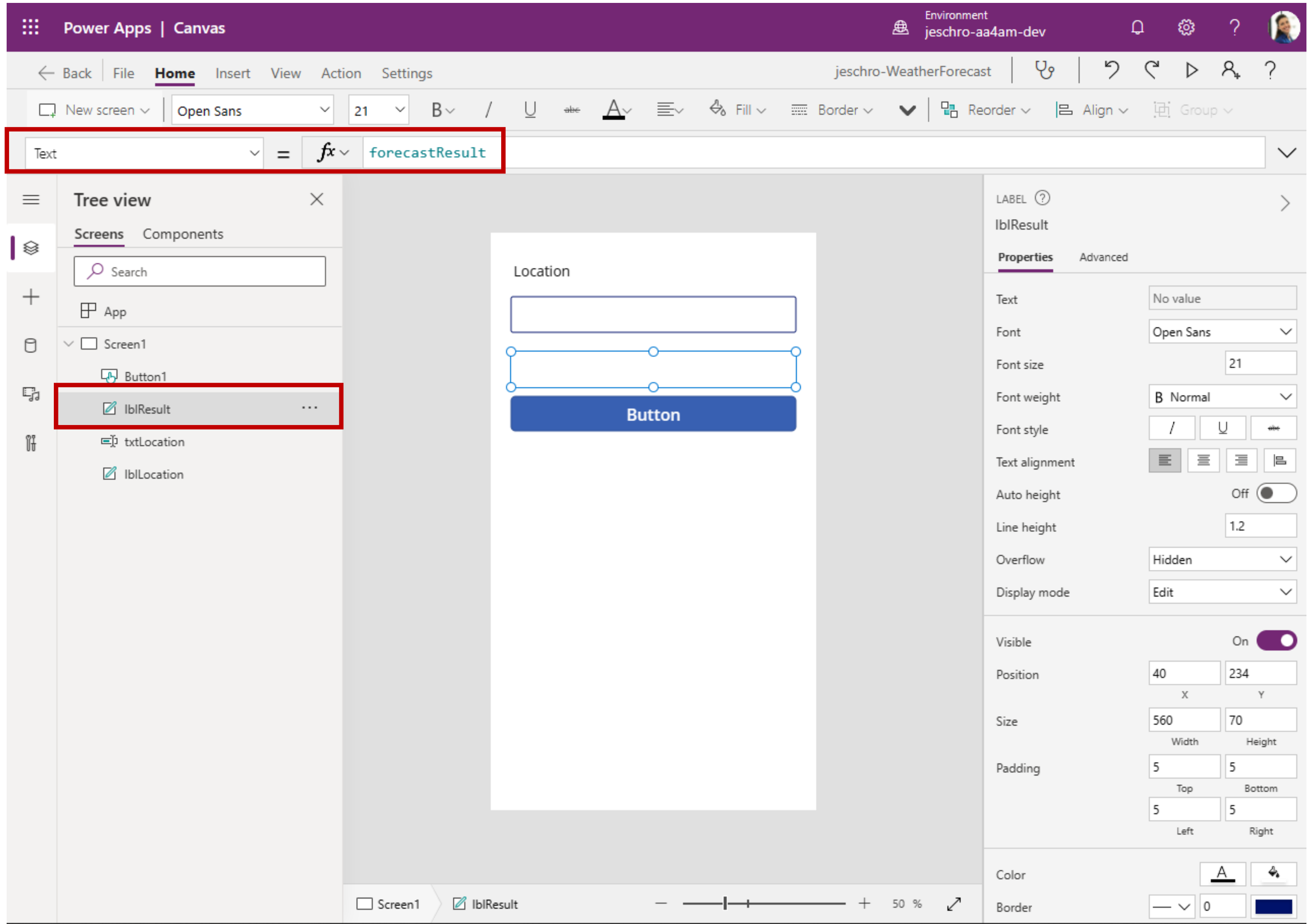
OnSelect = `Set(forecastResult, WeatherConnector.GetWeatherByLocation(txtLocation.Text).forecast)`

| | |
|---|---|
| Text | Button |
| Display mode | Edit |
| Visible | On |
| Position | X 40, Y 322 |
| Size | Width 560, Height 68 |
| Padding | Top 5, Bottom 5, Left 5, Right 5 |
| Border | 2 |
| Border radius | 10 |
| Font | Open Sans |
| Font size | 24 |
| Font weight | Semibold |
| Vertical align | Middle |

# Step 39. Display the result in the result label

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

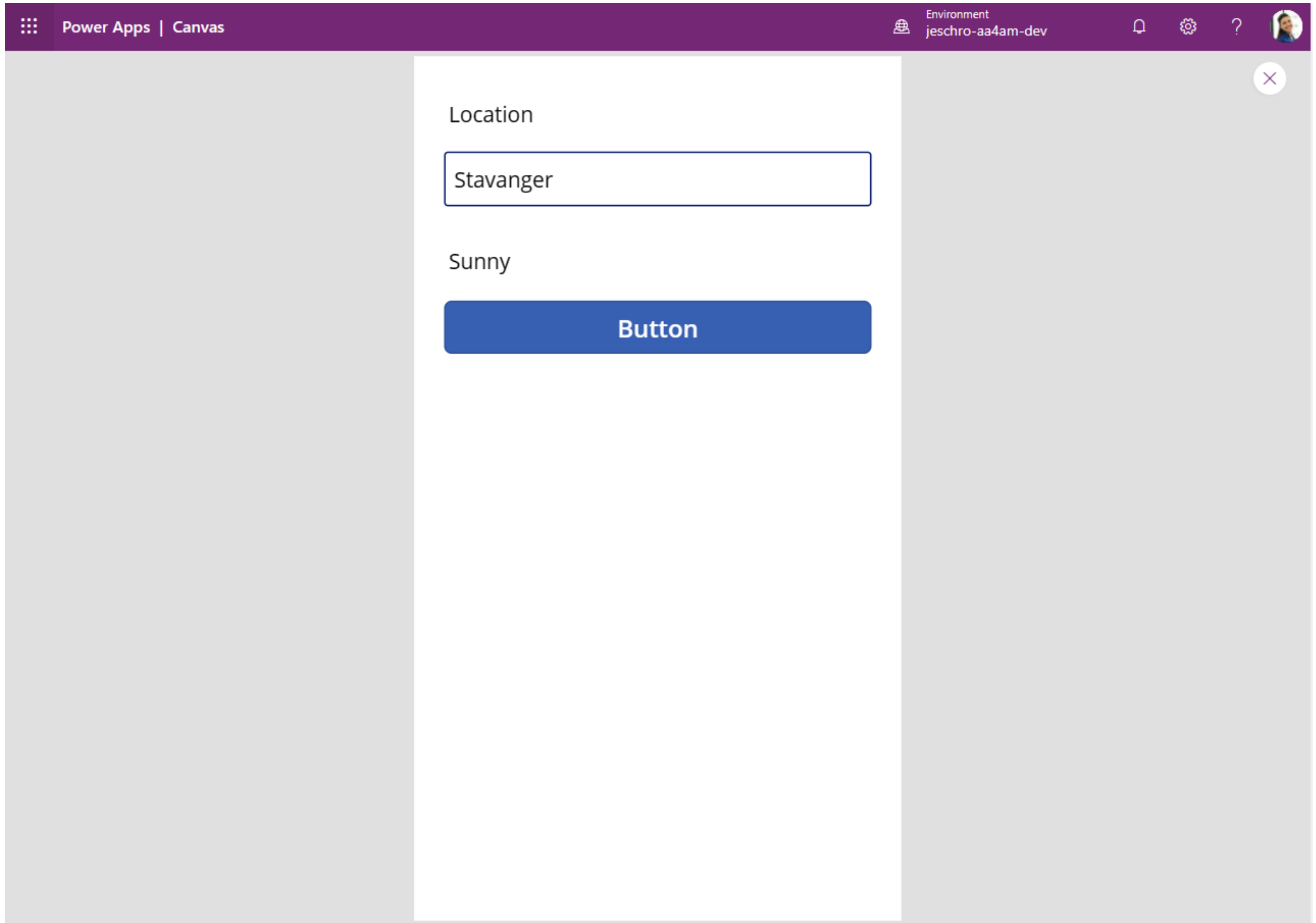
The label will show the value of the forecastResult variable


Text = forecastResult

## 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



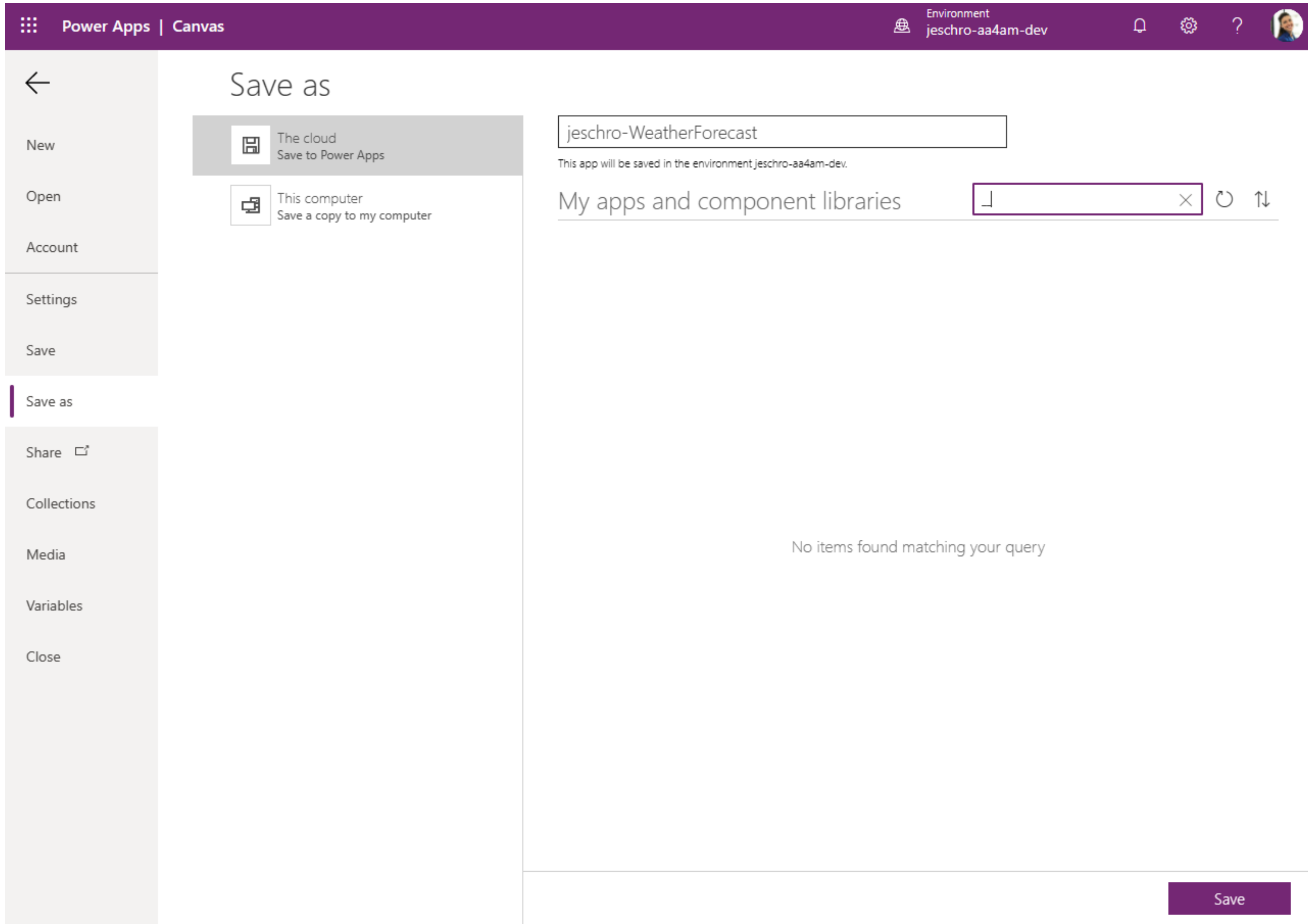
## Step 41. Save your app

Close the app player in top right corner.

Click **File => Save**

to save your app

You 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.

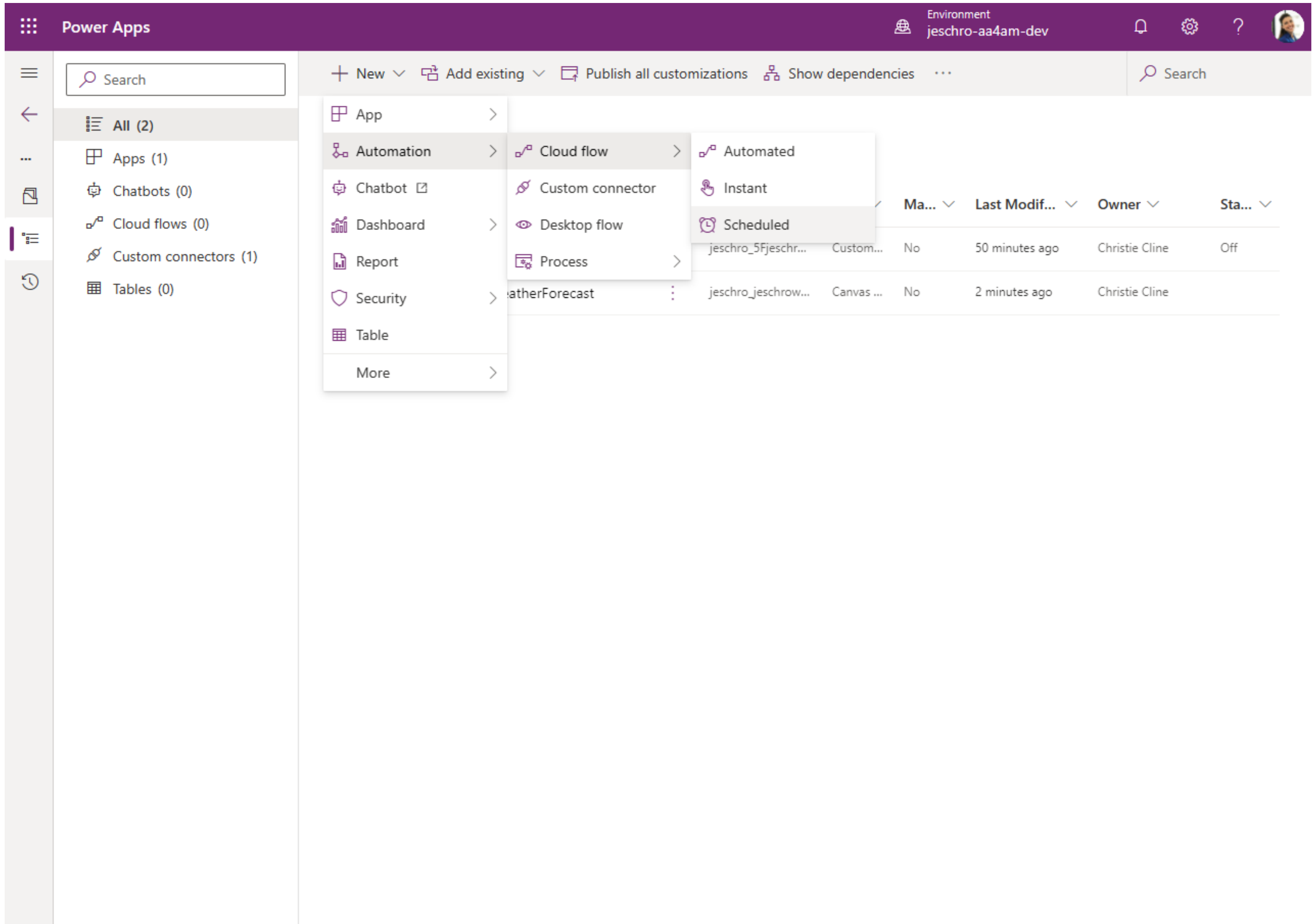


jeschro-WeatherForecast

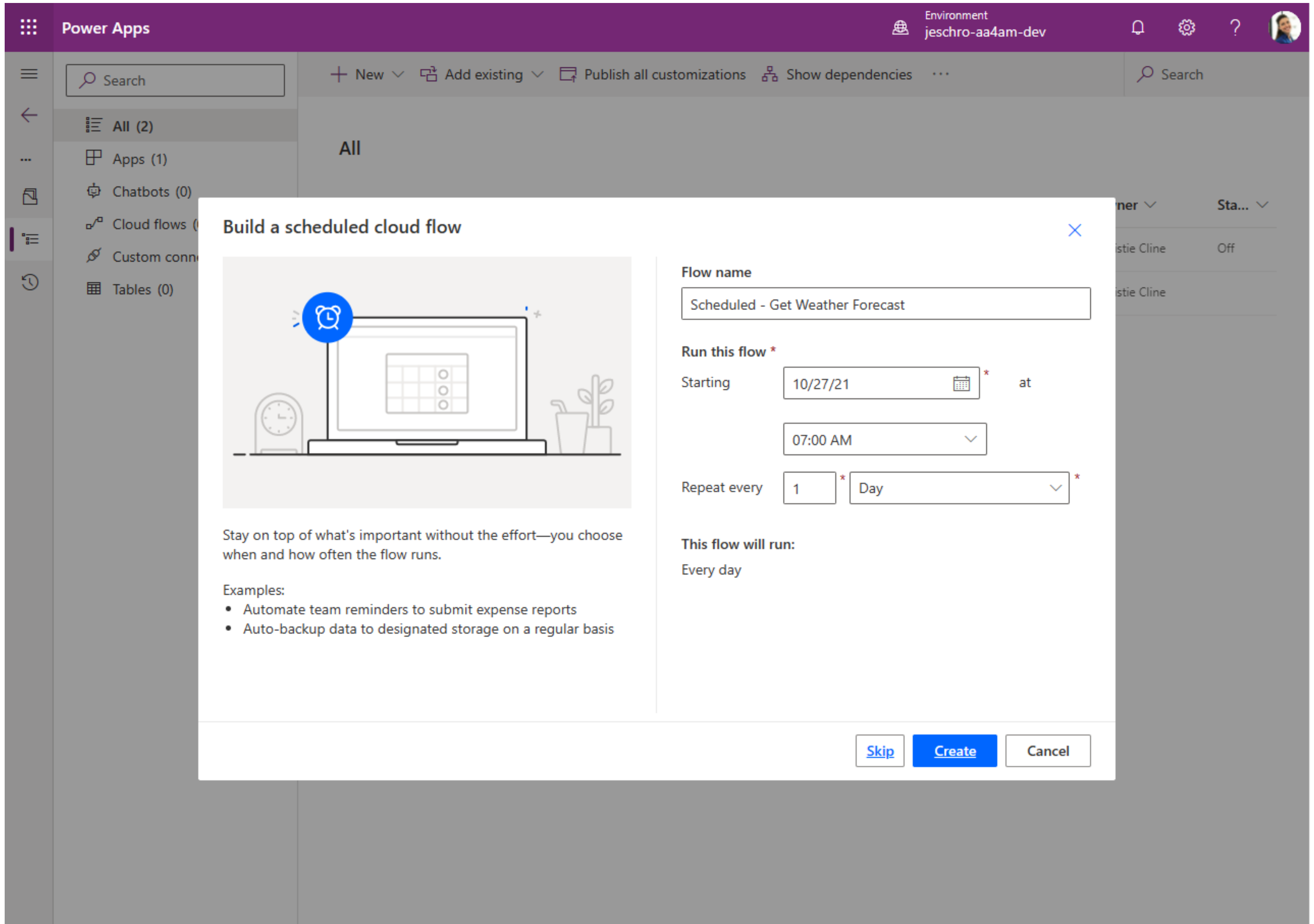
This app will be saved in the environment jeschro-aa4am-dev.

No items found matching your query

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



| | | | Ma... | Last Modif... | Owner | Sta... |
|---|---|---|---|---|---|---|
| | jeschro_5Fjeschr... | Custom... | No | 50 minutes ago | Christie Cline | Off |
| atherForecast | jeschro_jeschrow... | Canvas ... | No | 2 minutes ago | Christie Cline | |



Build a scheduled cloud flow

Stay on top of what's important without the effort—you choose when and how often the flow runs.

Examples:
- Automate team reminders to submit expense reports
- Auto-backup data to designated storage on a regular basis

Flow name: Scheduled - Get Weather Forecast

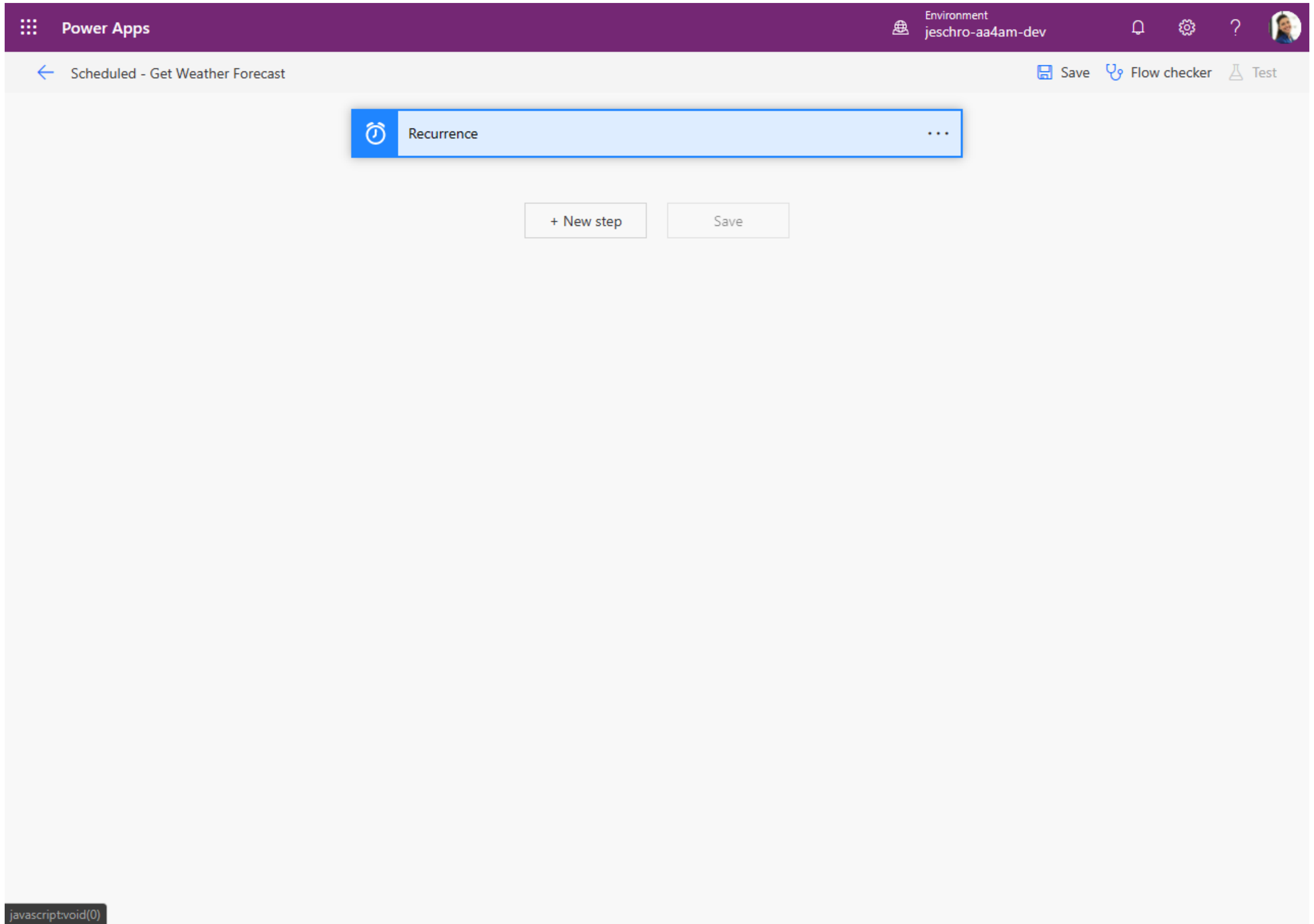
Run this flow *

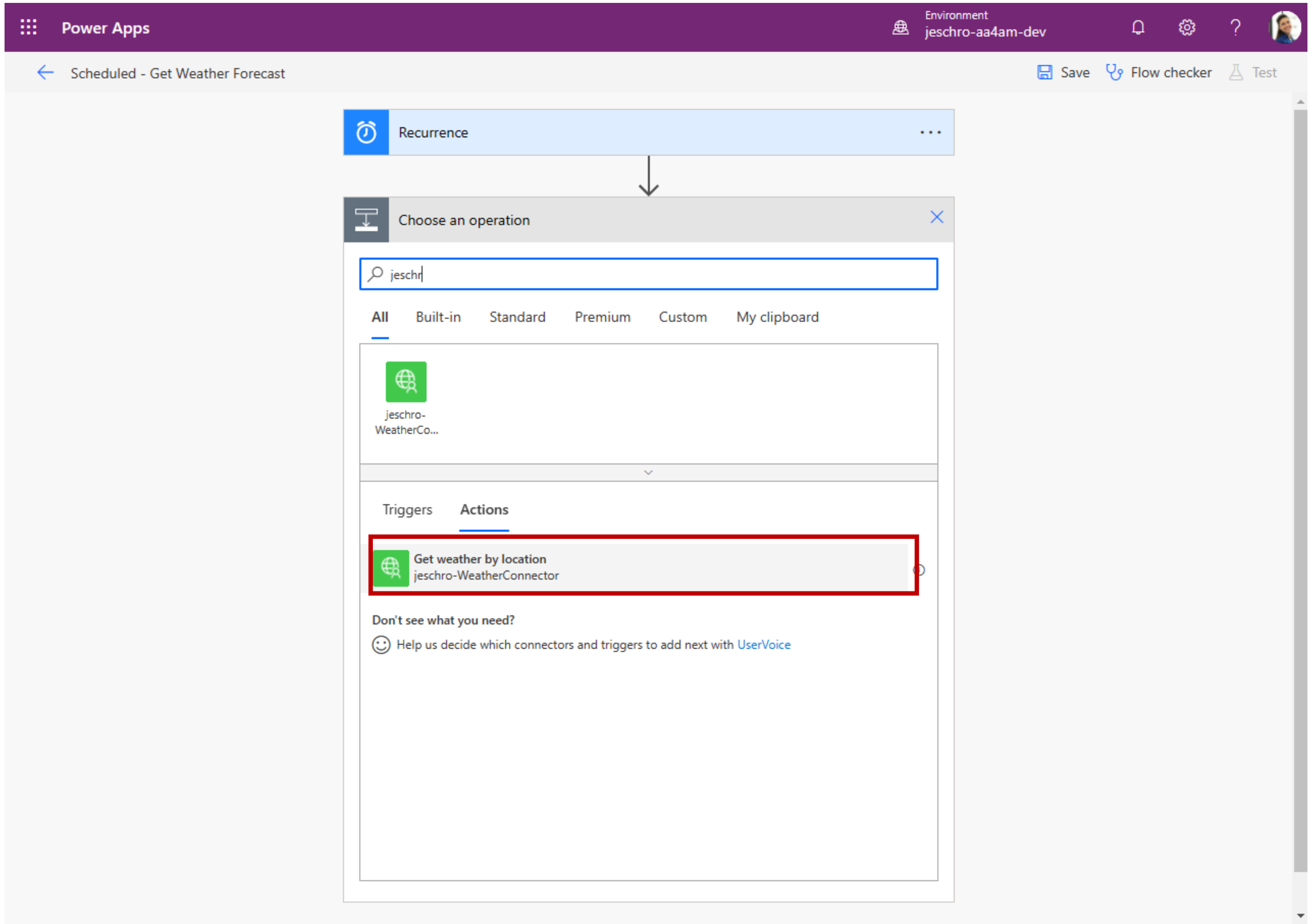
Starting: 10/27/21 * at 07:00 AM

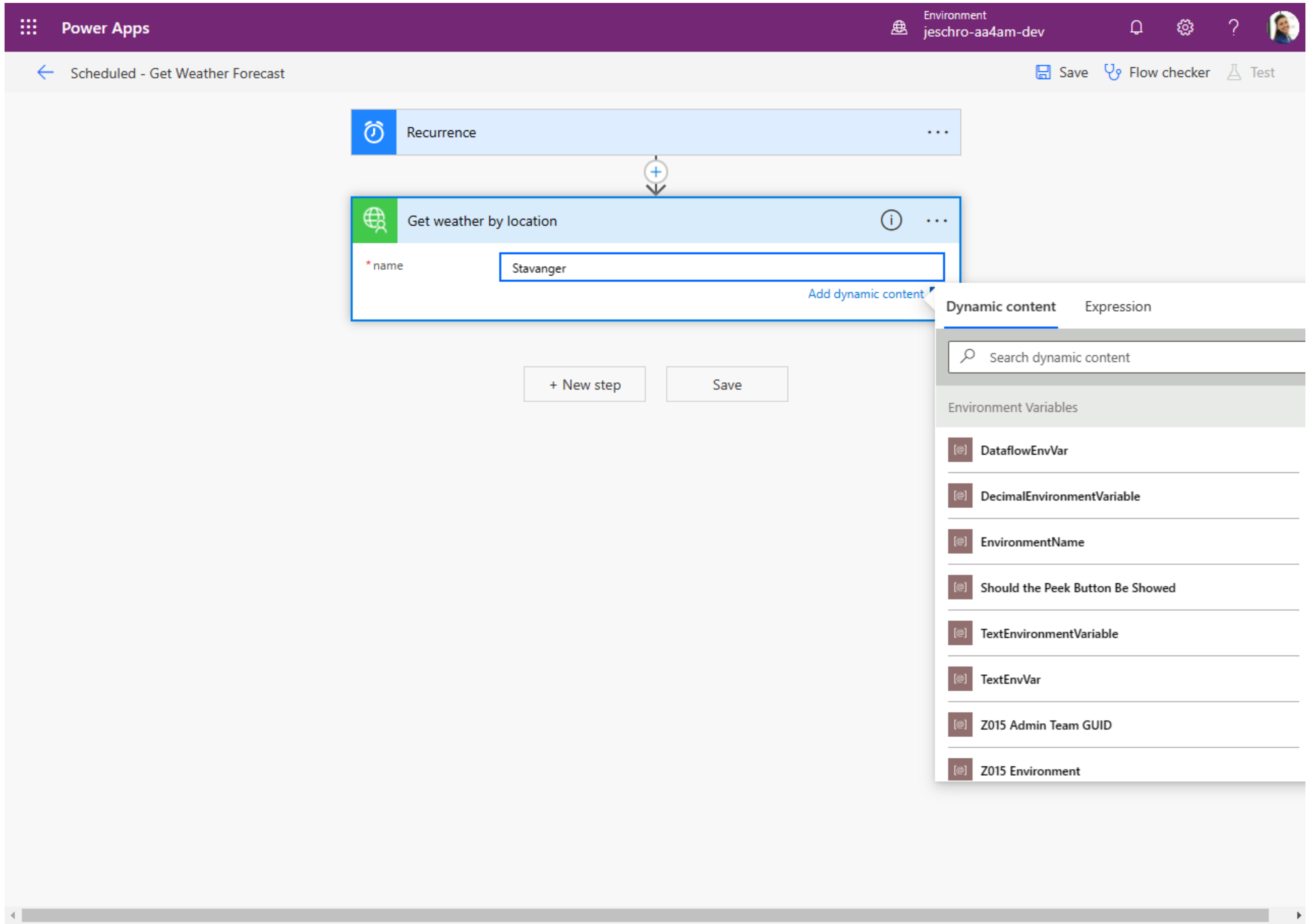
Repeat every: 1 * Day *

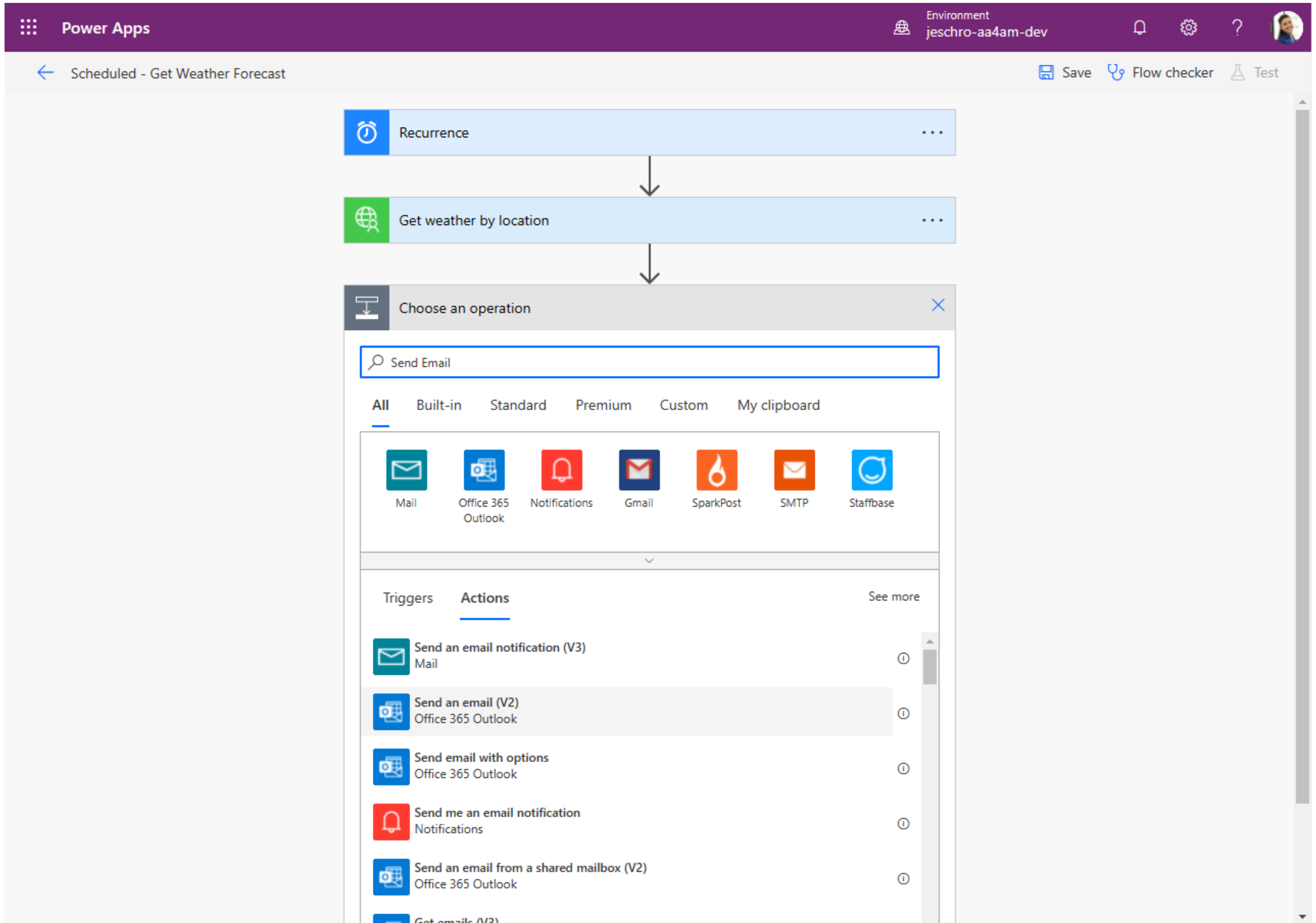
This flow will run:

Every day


Power Apps
Environment
jeschro-aa4am-dev
Scheduled - Get Weather Forecast
Save
Flow checker
Test
Recurrence
+ New step
Save
javascript:void(0)


Power Apps
Environment
jeschro-aa4am-dev
Scheduled - Get Weather Forecast
Save
Flow checker
Test
Recurrence
Choose an operation
jeschr
All
Built-in
Standard
Premium
Custom
My clipboard
jeschro-WeatherCo...
Triggers
Actions
Get weather by location
jeschro-WeatherConnector
Don't see what you need?
Help us decide which connectors and triggers to add next with UserVoice


Power Apps
Environment
jeschro-aa4am-dev
Scheduled - Get Weather Forecast
Save
Flow checker
Test
Recurrence
Get weather by location
* name
Stavanger
Add dynamic content
+ New step
Save
Dynamic content
Expression
Search dynamic content
Environment Variables
DataflowEnvVar
DecimalEnvironmentVariable
EnvironmentName
Should the Peek Button Be Showed
TextEnvironmentVariable
TextEnvVar
Z015 Admin Team GUID
Z015 Environment



Scheduled - Get Weather Forecast

Recurrence

Get weather by location

Choose an operation

Send Email

Send an email notification (V3)
Mail

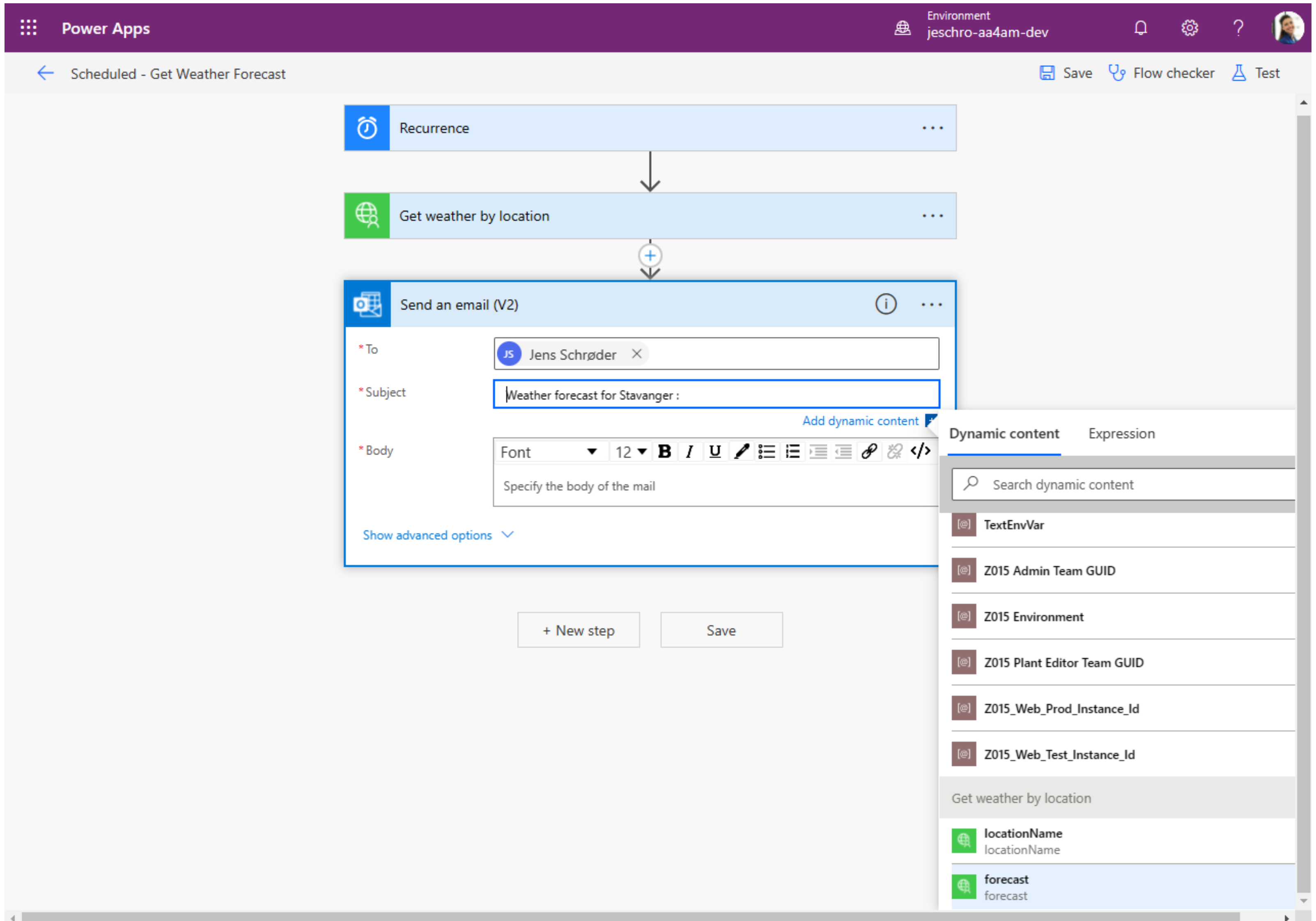
Send an email (V2)
Office 365 Outlook

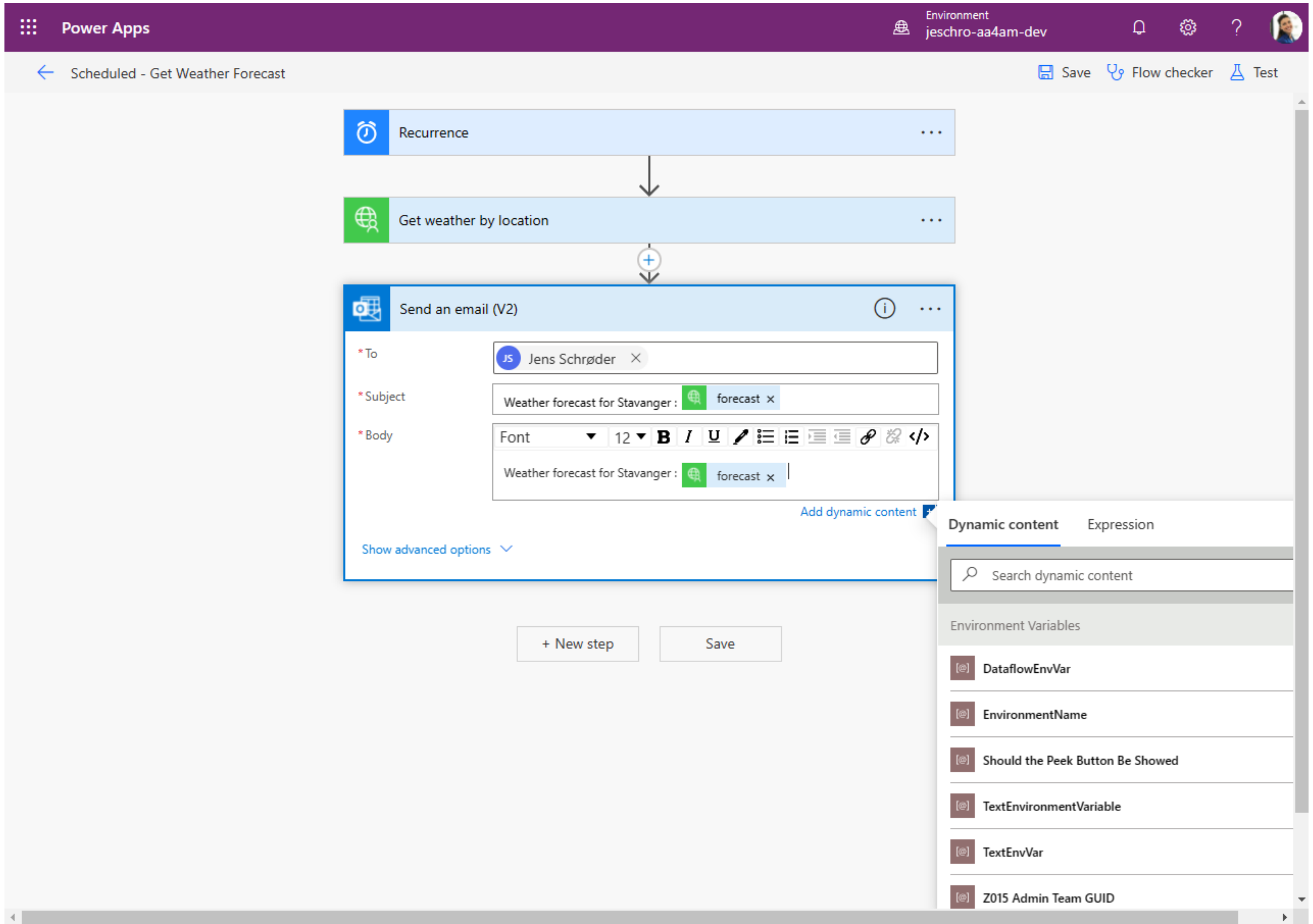
Send email with options
Office 365 Outlook

Send me an email notification
Notifications

Send an email from a shared mailbox (V2)
Office 365 Outlook

Get emails (V3)


Power Apps
Environment
jeschro-aa4am-dev
Scheduled - Get Weather Forecast
Save
Flow checker
Test
Recurrence
Get weather by location
Send an email (V2)
* To
Jens Schrøder
* Subject
Weather forecast for Stavanger :
Add dynamic content
* Body
Font
12
Specify the body of the mail
Show advanced options
+ New step
Save
Dynamic content
Expression
Search dynamic content
TextEnvVar
Z015 Admin Team GUID
Z015 Environment
Z015 Plant Editor Team GUID
Z015_Web_Prod_Instance_Id
Z015_Web_Test_Instance_Id
Get weather by location
locationName
locationName
forecast
forecast



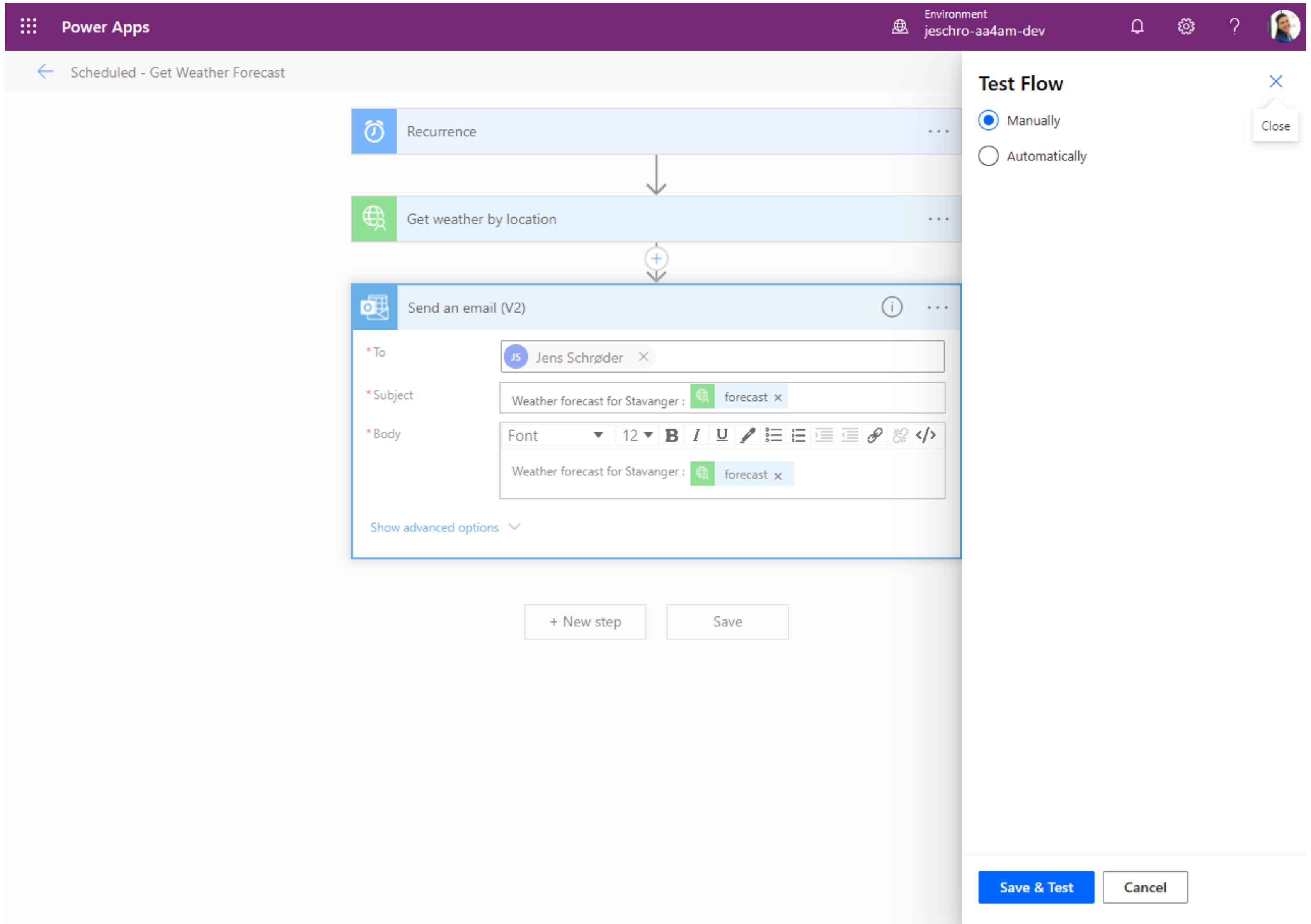
To: Jens Schrøder

Subject: Weather forecast for Stavanger : forecast

Body: Weather forecast for Stavanger : forecast

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

# Save and Test



Test Flow

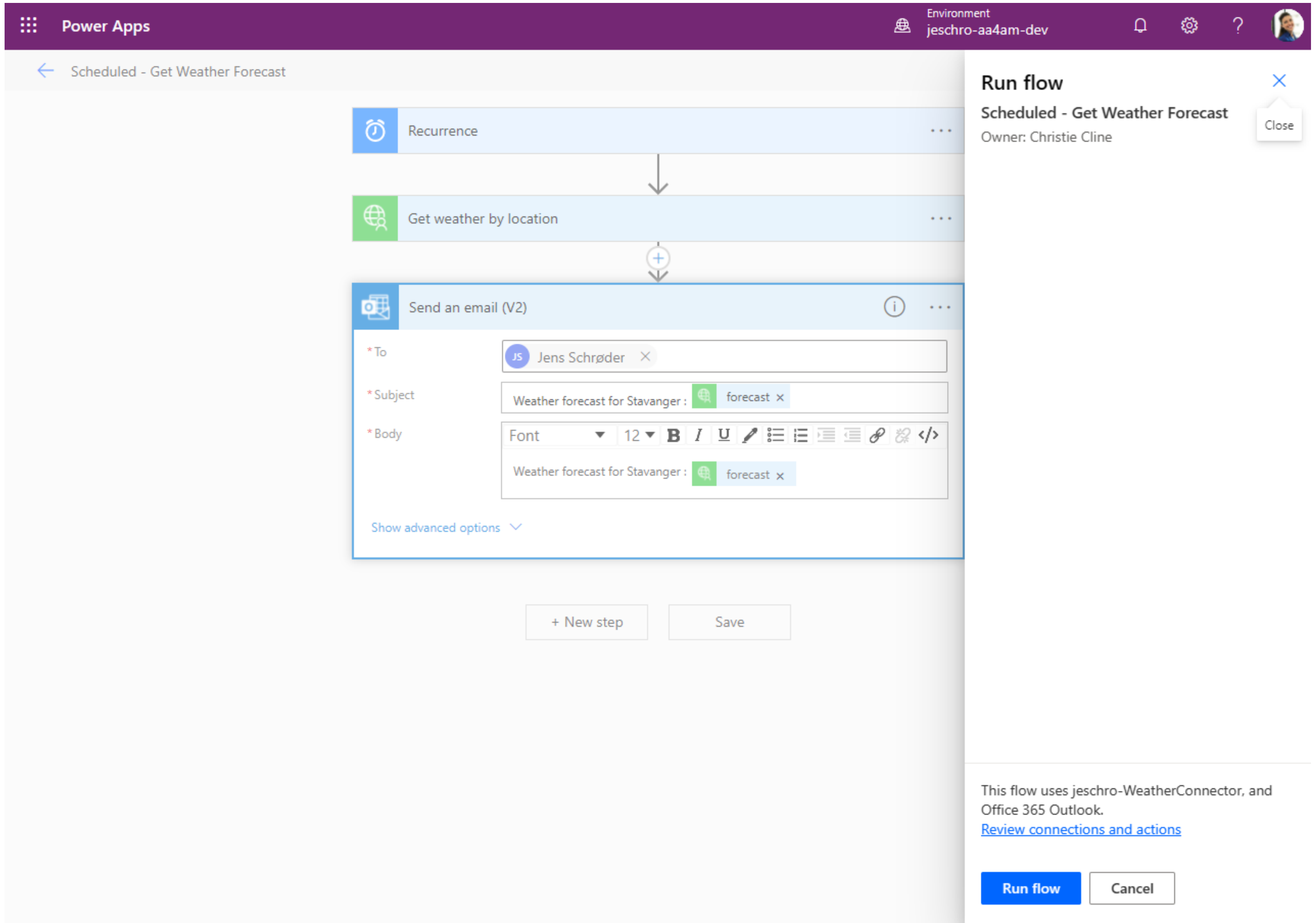
[✓ Manually] Automatically

*To: Jens Schrøder

*Subject: Weather forecast for Stavanger : forecast

*Body: Font 12

Weather forecast for Stavanger : forecast



Scheduled - Get Weather Forecast

*To: Jens Schrøder

*Subject: Weather forecast for Stavanger : forecast

*Body: Weather forecast for Stavanger : forecast

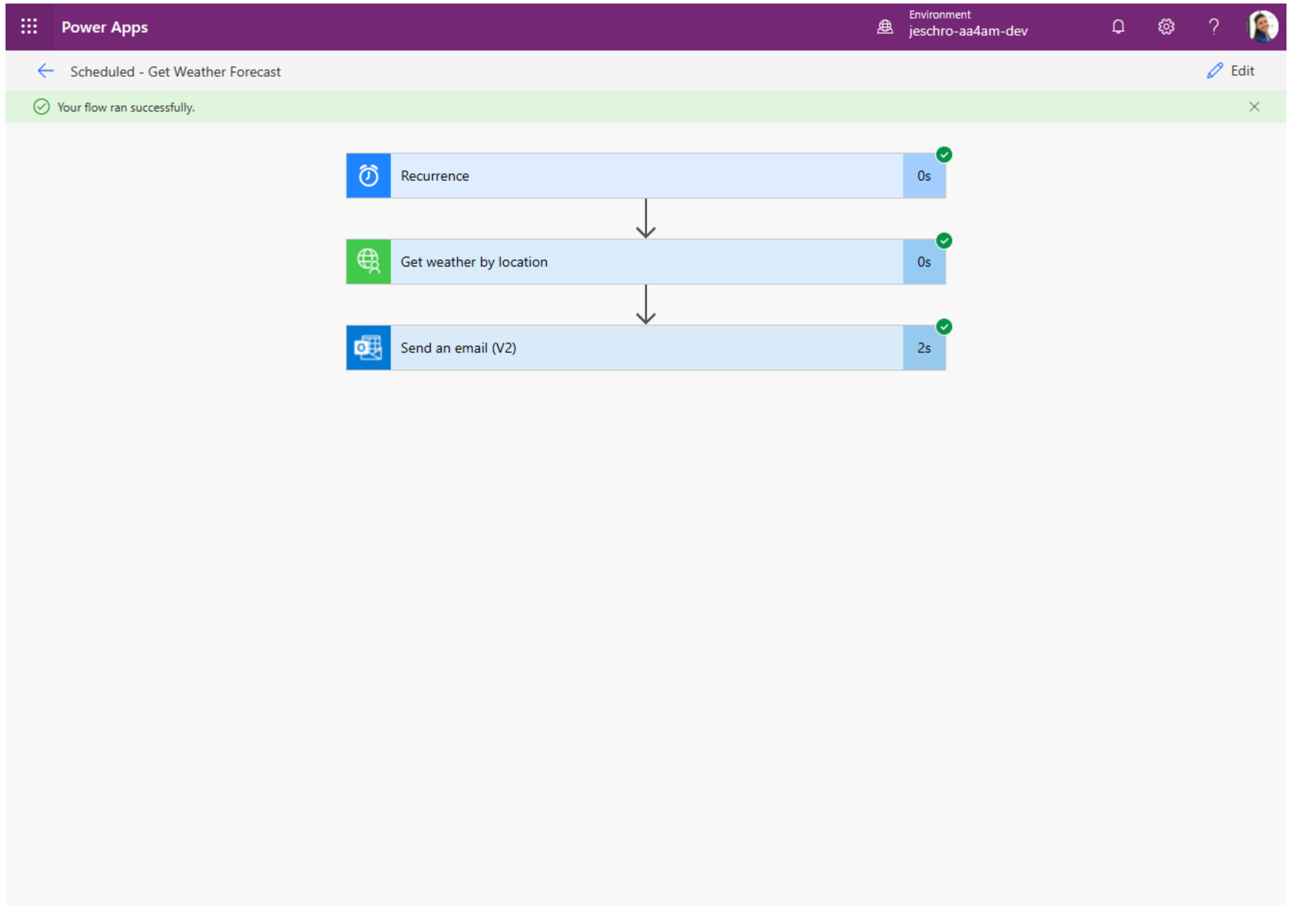
Run flow

Scheduled - Get Weather Forecast

Owner: Christie Cline

This flow uses jeschro-WeatherConnector, and Office 365 Outlook.

Review connections and actions

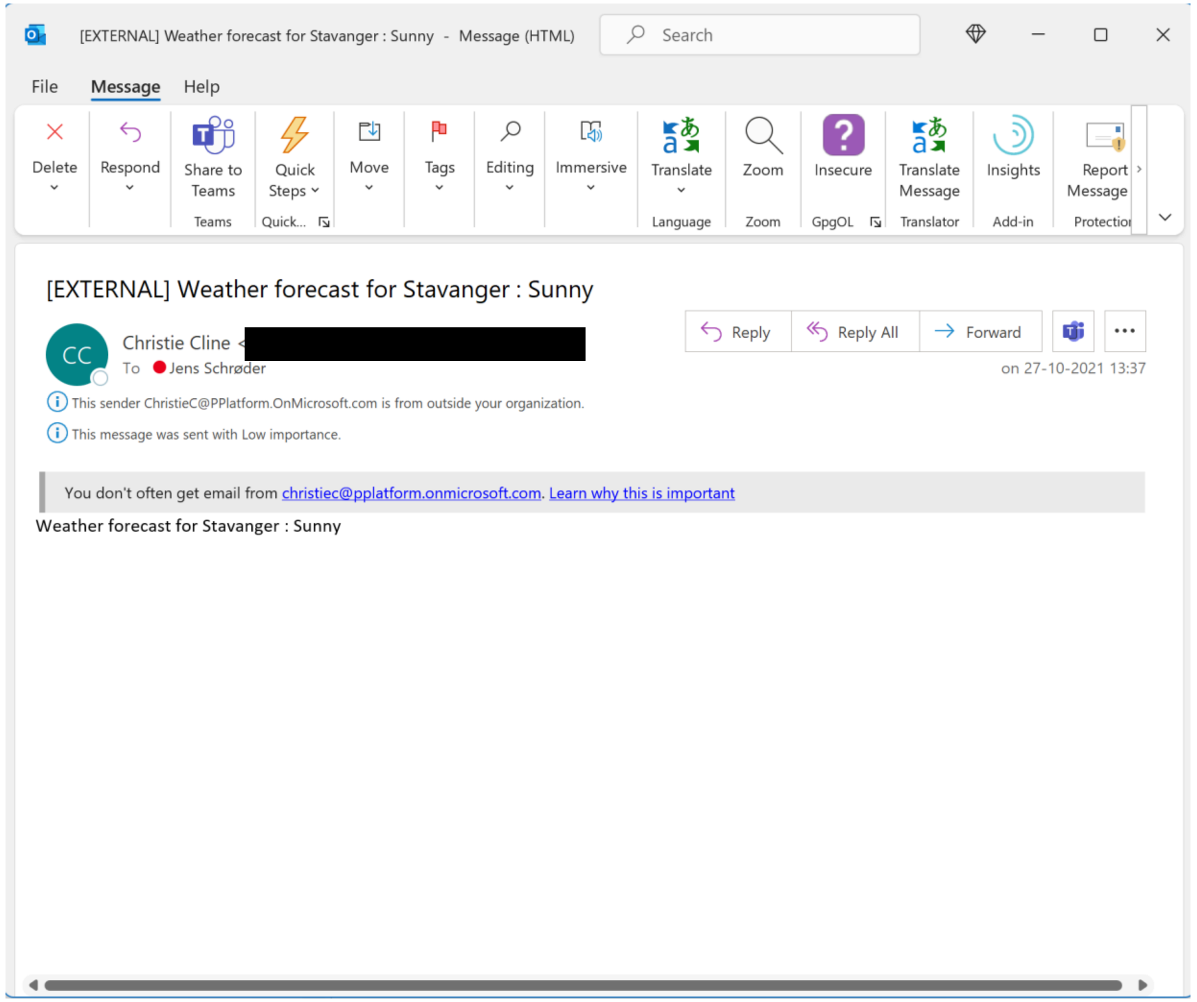

Scheduled - Get Weather Forecast

Your flow ran successfully.

Recurrence 0s

Get weather by location 0s

Send an email (V2) 2s


# [EXTERNAL] Weather forecast for Stavanger : Sunny

Christie Cline [redacted]

To Jens Schrøder

on 27-10-2021 13:37

This sender ChristieC@PPlatform.OnMicrosoft.com is from outside your organization.

This message was sent with Low importance.

You don't often get email from christiec@pplatform.onmicrosoft.com. Learn why this is important

Weather forecast for Stavanger : Sunny

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

How To: Use Swagger in Azure Functions - Cloudkasten