A close up of a sign

Description generated with high confidence

**Azure Functions and DevOps**

*Steef-jan Wiggers  
@steefJan*

Contents

[Objective 3](#_Toc4618971)

[Prerequisites 3](#_Toc4618972)

[Summary 3](#_Toc4618973)

[Step 1 - Create an Azure DevOps Project 4](#_Toc4618974)

[Step 2 - Create an Azure Function and test locally 6](#_Toc4618975)

[Step 3 - Add Unit tests and run the test project 12](#_Toc4618976)

[Step 4 – Push the project to Azure DevOps 22](#_Toc4618977)

[Step 5 - Create and execute a build template 24](#_Toc4618978)

[Step 6 - Create and execute a release template 29](#_Toc4618979)

# Objective

In this lab, you will familiarize yourself with Azure DevOps Build and Release Templates. building an Azure Function locally, pushing the code to a respository, creating and executing a build template, and doing a release.

## Prerequisites

* Azure Subscription
* Azure DevOps

## Summary

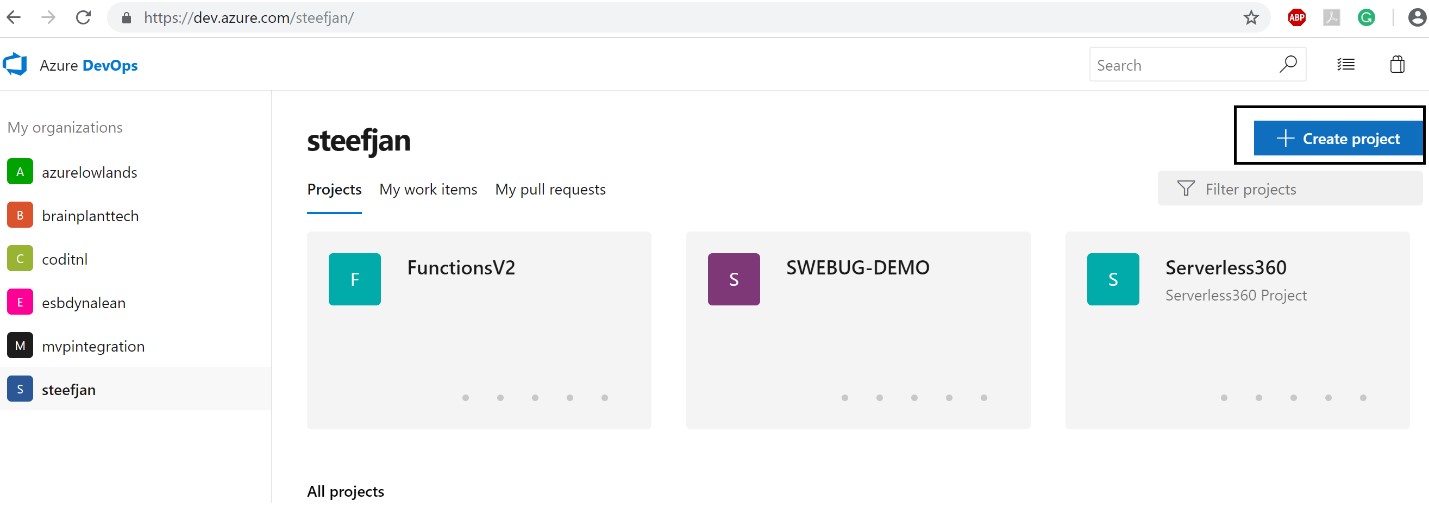
To build the solution in this lab, you have to follow the steps described in this section. From a high-level view the steps are:

* Create a project in Azure DevOps
* Create an Azure Function and test the Function Locally
* Add Unittests and run the test project
* Push code to Azure DevOps
* Create and execute a build template
* Create and execute a release template Lab duration: 45 minutes.

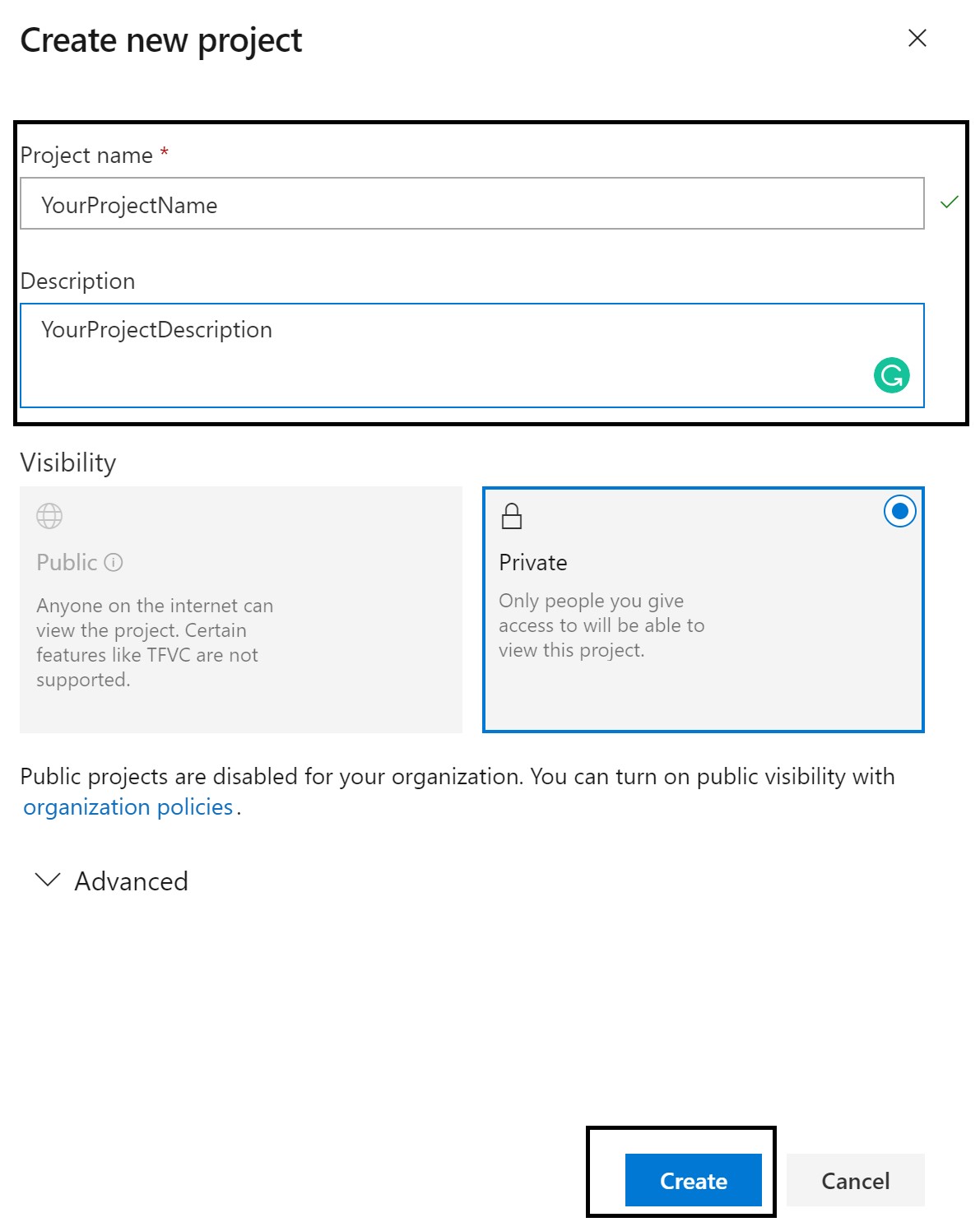
# Step 1 - Create an Azure DevOps Project

The very first step in this lab is

1. Go to [https://azure.microsoft.com/en-us/services/devops/.](https://azure.microsoft.com/en-us/services/devops/)
2. Create an account or sign in.
3. In the top right corner click + **Create Project**.



1. A new pane will appear.
2. Fill a descriptive name and description and click **Create**.

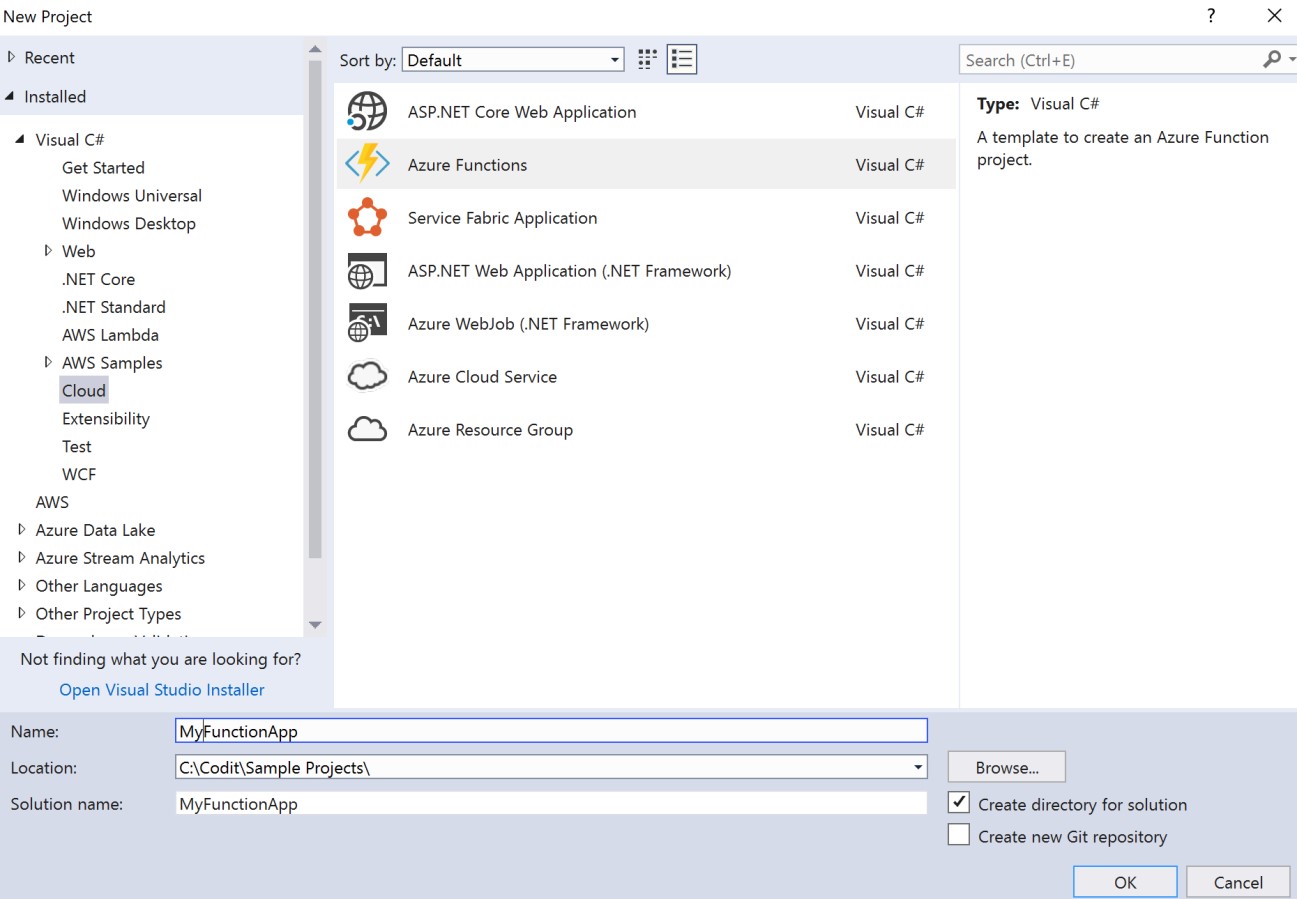


1. You have now a project in Azure DevOps.

# Step 2 - Create an Azure Function and test locally

In this step we will create a Function in Visual Studio 2017.

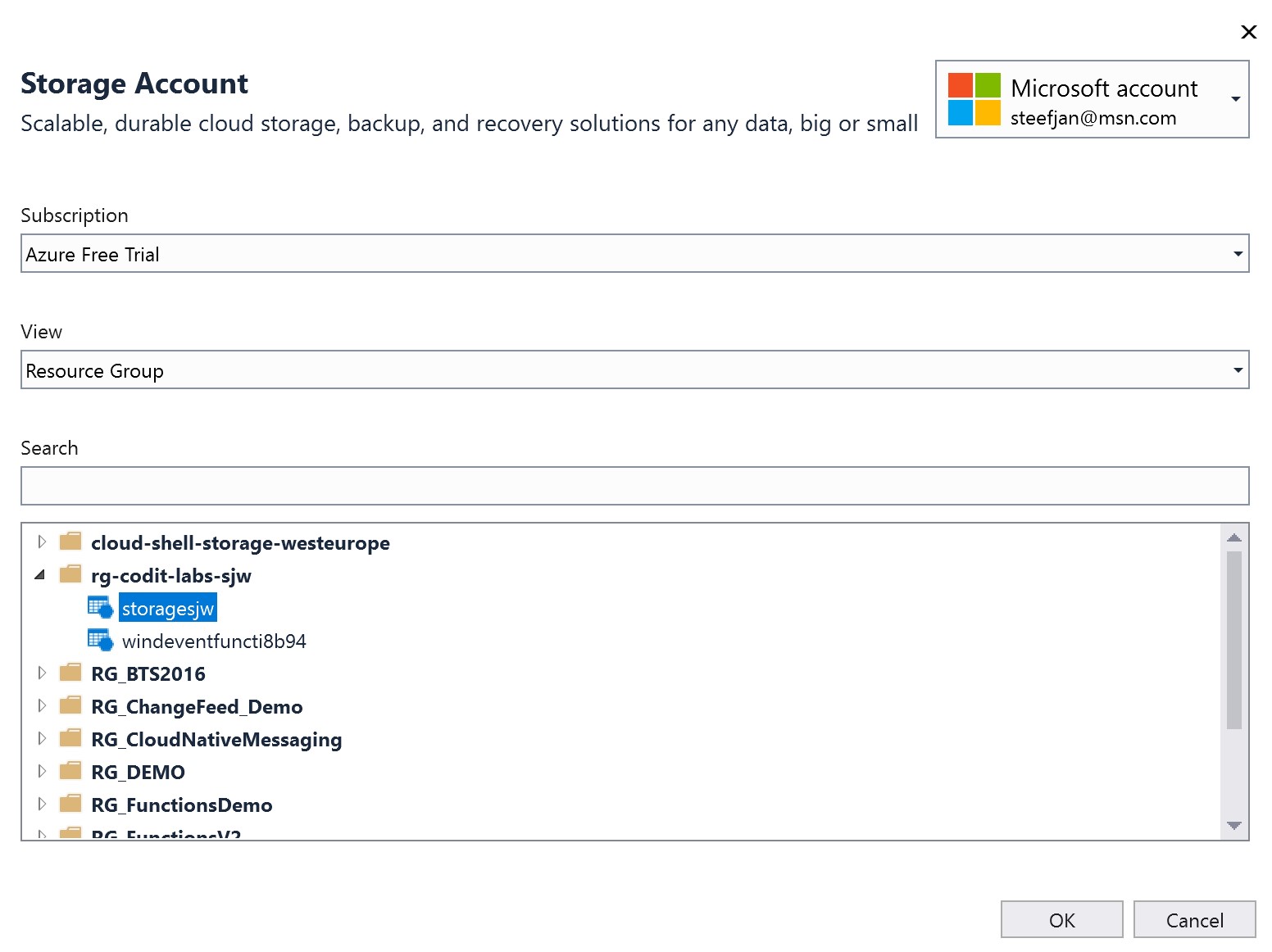
1. Open Visual Studio 2017.
2. Select **File** --> **New** --> **Project**
3. In the New Project Pane, select **Cloud**.
4. Subsequently, choose **Azure Functions**.
5. Provide a name, choose the correct path, and solution name.



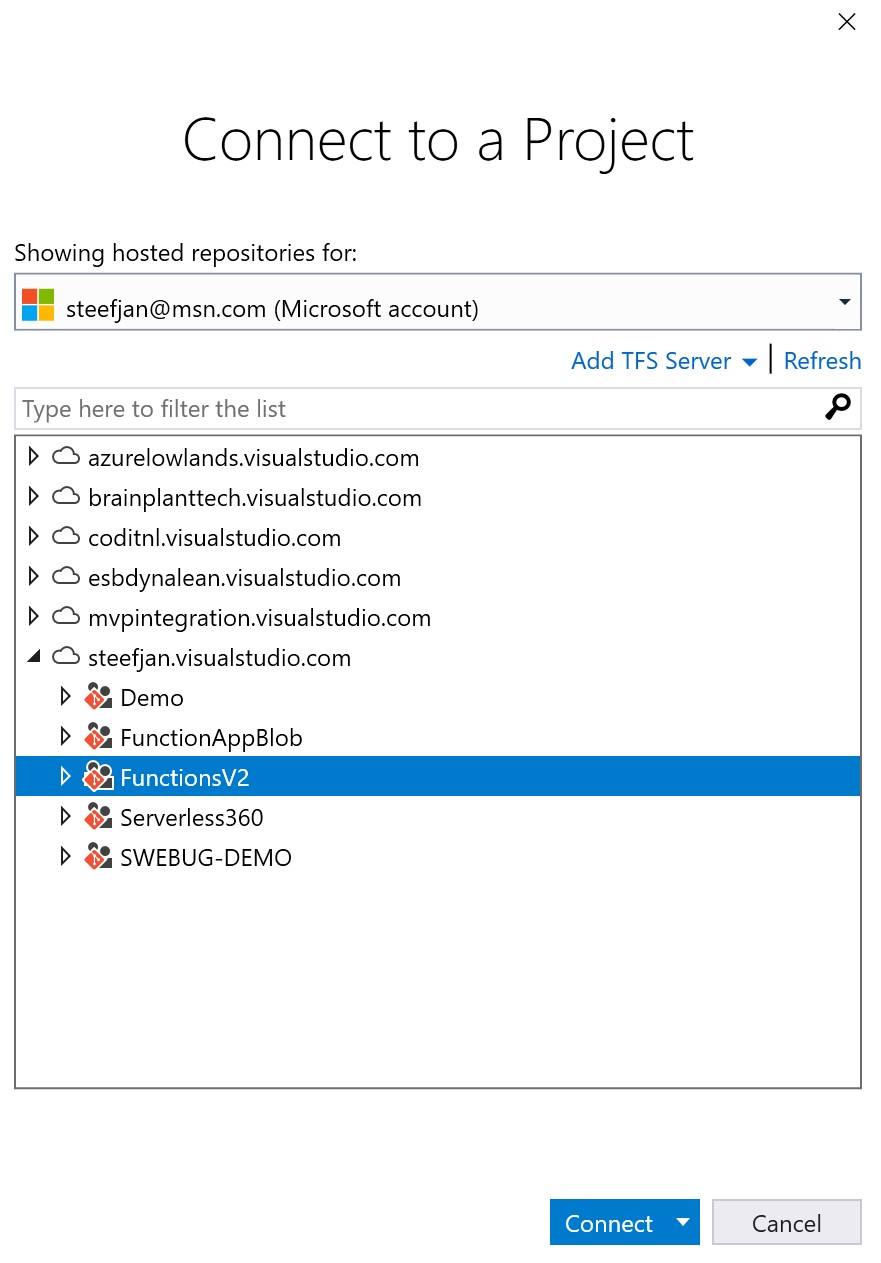
1. Click **Ok**.
2. Choose **Azure Functions V2**.

Note: Choose the .NET Standard 2.0, which is compatible with .NET Core 2 (Function V2). This framework works with well in conjunction with Framework 4.6.1 you will use/require for the unti test project!

1. For Storage Account choose ***Browse***.
2. Pick a storage account from earlier lab or create a new one.



1. Click **Ok**.
2. Choose the **HTTP Trigger Template** and click **Ok**.
3. Click **Team Explorer** on the right side of Visual Studio.
4. Click manage connections.
5. Click Connect to a project.
6. It will show the repositories belonging to your account.
7. Select the project you created in step 1 and click **Connect**.



1. Right click solution and choose add to source control.
2. Rename function1.cs to ConvertWindSpeedToBeaufort.cs.
3. Copy the following code/past the following code above your namespace

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Azure.WebJobs;

using Microsoft.Azure.WebJobs.Extensions.Http;

using Microsoft.Azure.WebJobs.Host;

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

1. Copy the following code below your namespace:

public static class ConvertWindSpeedToBeaufort

{

[FunctionName("ConvertWindSpeedToBeaufort")]

public static IActionResult Run([HttpTrigger(AuthorizationLevel.Function, "get", "post", Route = null)]HttpRequest req, TraceWriter log)

{

var json = new StreamReader(req.Body).ReadToEnd();

WindSpeedData data = null;

var content = string.Empty;

try

{

data = JsonConvert.DeserializeObject<WindSpeedData>(json);

//Initalize beaufort scale

var beaufort = 0;

var cases = new Dictionary<Func<double, bool>, Action>

{

{ x => x < 0.2 , () => beaufort = 0} ,

{ x => x < 1.5 , () => beaufort = 1} ,

{ x => x < 3.3 , () => beaufort = 2} ,

{ x => x < 5.4 , () => beaufort = 3} ,

{ x => x < 7.9 , () => beaufort = 4} ,

{ x => x < 10.7, () => beaufort = 5} ,

{ x => x < 13.8, () => beaufort = 6} ,

{ x => x < 17.1, () => beaufort = 7} ,

{ x => x < 20.7, () => beaufort = 8} ,

{ x => x < 24.4, () => beaufort = 9} ,

{ x => x < 28.4, () => beaufort = 10} ,

{ x => x < 32.6, () => beaufort = 11} ,

{ x => x > 32.7, () => beaufort = 12}

};

cases.First(kvp => kvp.Key(data.WindSpeed)).Value();

data.Beaufort = beaufort;

content = JsonConvert.SerializeObject(data, Formatting.Indented);

log.Info($"Response messagebody : " + content);

}

catch (Exception e)

{

data = null;

log.Info($"Function call error message : " + e.Message);

}

log.Info($"Function call with Request messagebody : " + json);

return data != null

? (ActionResult)new OkObjectResult(content)

: new BadRequestObjectResult("Please pass the correct request body!");

}

}

1. Add a class to the project and name it WindSpeedData.cs 22. Copy the following code below the namesapce:

public class WindSpeedData

{

public string Location { get; set; }

public double WindSpeed { get; set; }

public int Beaufort { get; set; }

}

1. Build the project.
2. Run the project.
3. Copy the endpoint <http://localhost:7071/api/ConvertWindSpeedToBeaufort>26. Open Postman, and create a POST to the URL
4. Use the following payload:

{

"Location": "Amsterdam",

"WindSpeed": 12,

"Beaufort": 0

}

1. Click Send.
2. You will get the following response:

{

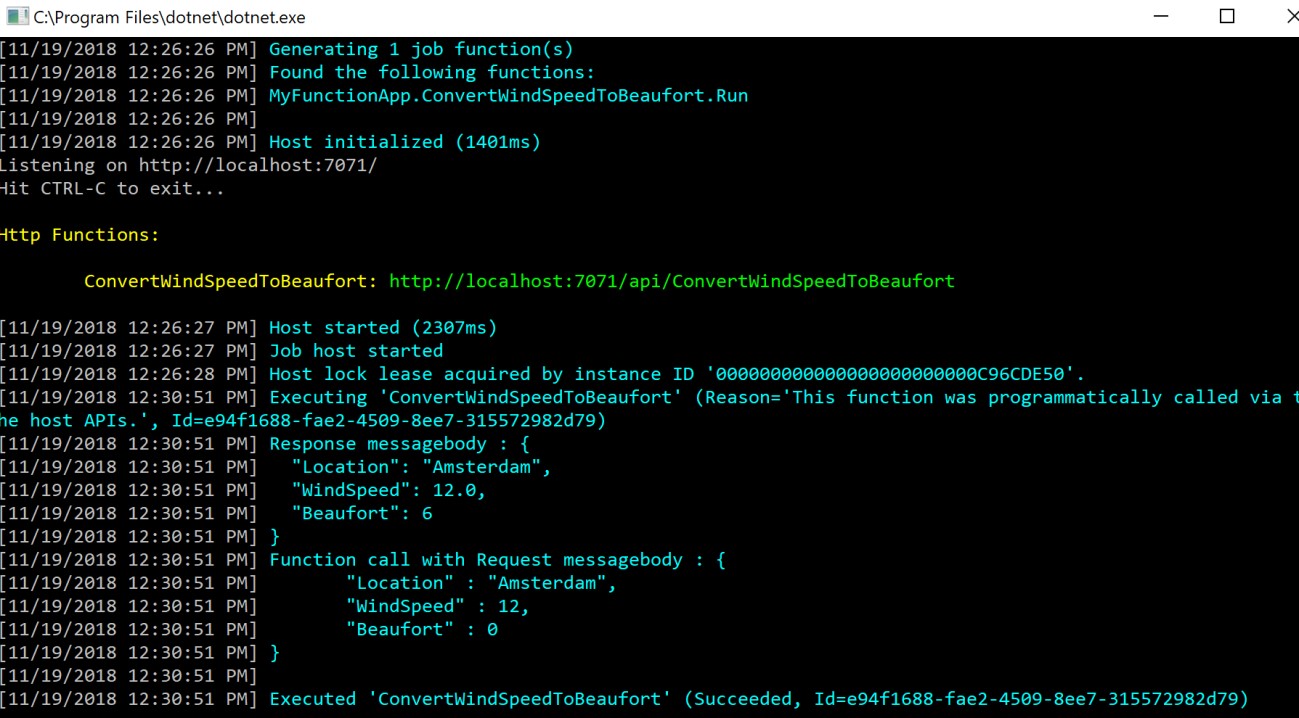
"Location": "Amsterdam",

"WindSpeed": 12.0,

"Beaufort": 6

}

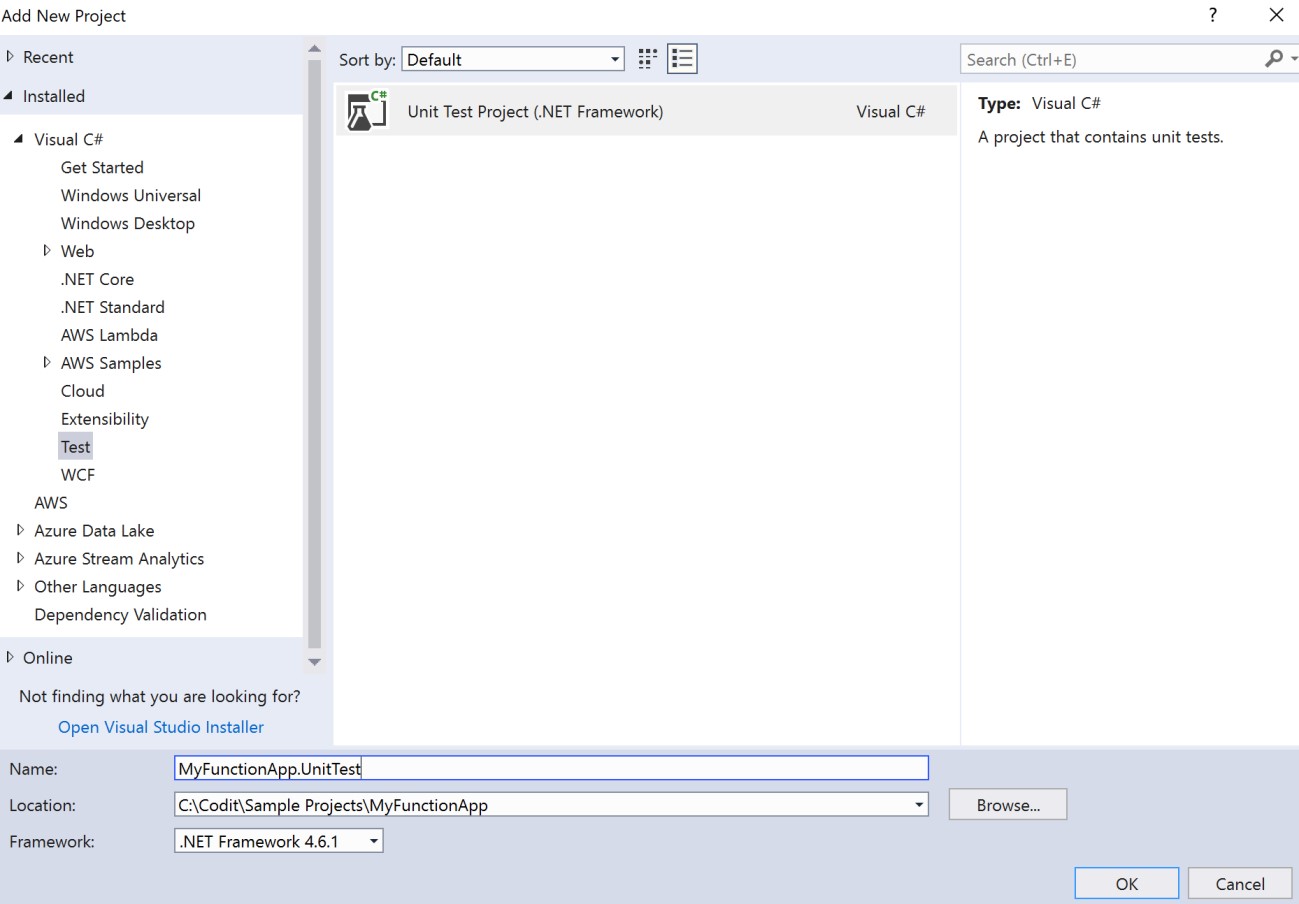
1. Observe the console window that fired up after running the function.



# Step 3 - Add Unit tests and run the test project

In this step we will add a unit test project and a couple of tests.

1. Add a project to the exsisting project you created in step 2 by right clicking the solution --> Add -> New Project
2. In the New Project Pane, select **Test**.
3. Choose **Unit Test Project**.
4. Specify a descriptive name, and choose the right location.



1. Click **Ok**.
2. Right Click the newly added test project and choose New Folder
3. Name it FunctionTestHelper
4. Rightclick the folder and add New Item.
5. Choose Class and name it FunctionTest.cs.
6. Past the following code above the namespace:

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Http.Internal;

using Microsoft.Azure.WebJobs;

using Microsoft.Azure.WebJobs.Host;

using Microsoft.Extensions.Primitives;

using Moq;

using System;

using System.Collections.Generic;

using System.IO;

using System.Threading;

using System.Threading.Tasks;

1. And the following code below the namespace:

public abstract class FunctionTest

{

protected TraceWriter log = new VerboseDiagnosticsTraceWriter();

public HttpRequest HttpRequestSetup(Dictionary<String, StringValues> query, string body)

{

var reqMock = new Mock<HttpRequest>();

reqMock.Setup(req => req.Query).Returns(new QueryCollection(query));

var stream = new MemoryStream();

var writer = new StreamWriter(stream);

writer.Write(body);

writer.Flush();

stream.Position = 0;

reqMock.Setup(req => req.Body).Returns(stream);

return reqMock.Object;

}

}

public class AsyncCollector<T> : IAsyncCollector<T>

{

public readonly List<T> Items = new List<T>();

public Task AddAsync(T item, CancellationToken cancellationToken = default(CancellationToken))

{

Items.Add(item);

return Task.FromResult(true);

}

public Task FlushAsync(CancellationToken cancellationToken = default(CancellationToken))

{

return Task.FromResult(true);

}

}

1. Rightclick the folder and add New Item.
2. Choose Class and name it FunctionTest.cs.
3. Above the namespace past the following code:

using Microsoft.Azure.WebJobs.Host;

using System.Diagnostics;

1. Below the namespace paste the following code:

public class VerboseDiagnosticsTraceWriter : TraceWriter

{

public VerboseDiagnosticsTraceWriter() : base(TraceLevel.Verbose)

{

}

public override void Trace(TraceEvent traceEvent)

{

Debug.WriteLine(traceEvent.Message);

}

}

1. Right click test project and choose manage nuget packages.
2. Find Castle.core and install it.
3. Next find Microsoft.AspNetCore and choose version 2.0.1 and install.
4. Next find Microsoft.Azure.WebJobs and choose version 3.0.0-beta5 (note in the search check the **include prereleases**) and install it.
5. Next, find Microsoft.AspNetCore.Mvc and choose version 2.0.2 and install it.
6. Lastly, find Moq and choose version 4.10 and install it.
7. Rename UnitTest1.cs to UnitTestConvertWindSpeedData.cs
8. Next copy the following code above the namespace:

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Primitives;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

1. And the following code below the namespace:

[TestClass]

public class UnitTestConvertWindSpeedData : FunctionTestHelper.FunctionTest

{

[TestMethod]

public void CanConvertLowWindSpeed()

{

//Arrange

var windSpeedRequest = new WindSpeedData

{

WindSpeed = 8.0,

Beaufort = 0,

Location = "Amsterdam"

};

//Act

var query = new Dictionary<String, StringValues>();

var body = JsonConvert.SerializeObject(windSpeedRequest, Formatting.Indented);

var result = ConvertWindSpeedToBeaufort.Run(HttpRequestSetup(query, body), log);

var resultObject = (OkObjectResult)result;

//Assert

var resultResponse = new WindSpeedData

{

WindSpeed = 8.0,

Beaufort = 5,

Location = "Amsterdam"

};

var resultBody = JsonConvert.SerializeObject(resultResponse, Formatting.Indented);

Assert.AreEqual(resultBody, resultObject.Value);

}

[TestMethod]

public void CanConvertWindSpeedToBeaufort3()

{

//Arrange

var windSpeedRequest = new WindSpeedData

{

WindSpeed = 4.0,

Beaufort = 0,

Location = "Amsterdam"

};

//Act

var query = new Dictionary<String, StringValues>();

var body = JsonConvert.SerializeObject(windSpeedRequest, Formatting.Indented);

var result = ConvertWindSpeedToBeaufort.Run(HttpRequestSetup(query, body), log);

var resultObject = (OkObjectResult)result;

//Assert

var resultResponse = new WindSpeedData

{

WindSpeed = 4.0,

Beaufort = 3,

Location = "Amsterdam"

};

var resultBody = JsonConvert.SerializeObject(resultResponse, Formatting.Indented);

Assert.AreEqual(resultBody, resultObject.Value);

}

[TestMethod]

public void CanConvertWindSpeedToBeaufort4()

{

//Arrange

var windSpeedRequest = new WindSpeedData

{

WindSpeed = 7.5,

Beaufort = 0,

Location = "Amsterdam"

};

//Act

var query = new Dictionary<String, StringValues>();

var body = JsonConvert.SerializeObject(windSpeedRequest, Formatting.Indented);

var result = ConvertWindSpeedToBeaufort.Run(HttpRequestSetup(query, body), log);

var resultObject = (OkObjectResult)result;

//Assert

var resultResponse = new WindSpeedData

{

WindSpeed = 7.5,

Beaufort = 4,

Location = "Amsterdam"

};

var resultBody = JsonConvert.SerializeObject(resultResponse, Formatting.Indented);

Assert.AreEqual(resultBody, resultObject.Value);

}

[TestMethod]

public void CanConvertWindSpeedToBeaufort6()

{

//Arrange

var windSpeedRequest = new WindSpeedData

{

WindSpeed = 12.0,

Beaufort = 0,

Location = "Amsterdam"

};

//Act

var query = new Dictionary<String, StringValues>();

var body = JsonConvert.SerializeObject(windSpeedRequest, Formatting.Indented);

var result = ConvertWindSpeedToBeaufort.Run(HttpRequestSetup(query, body), log);

var resultObject = (OkObjectResult)result;

//Assert

var resultResponse = new WindSpeedData

{

WindSpeed = 12.0,

Beaufort = 6,

Location = "Amsterdam"

};

var resultBody = JsonConvert.SerializeObject(resultResponse, Formatting.Indented);

Assert.AreEqual(resultBody, resultObject.Value);

}

[TestMethod]

public void CanConvertWindSpeedToBeaufort7()

{

//Arrange

var windSpeedRequest = new WindSpeedData

{

WindSpeed = 15.0,

Beaufort = 0,

Location = "Amsterdam"

};

//Act

var query = new Dictionary<String, StringValues>();

var body = JsonConvert.SerializeObject(windSpeedRequest, Formatting.Indented);

var result = ConvertWindSpeedToBeaufort.Run(HttpRequestSetup(query, body), log);

var resultObject = (OkObjectResult)result;

//Assert

var resultResponse = new WindSpeedData

{

WindSpeed = 15.0,

Beaufort = 7,

Location = "Amsterdam"

};

var resultBody = JsonConvert.SerializeObject(resultResponse, Formatting.Indented);

Assert.AreEqual(resultBody, resultObject.Value);

}

[TestMethod]

public void CanConvertWindSpeedToBeaufort8()

{

//Arrange

var windSpeedRequest = new WindSpeedData

{

WindSpeed = 19.0,

Beaufort = 0,

Location = "Amsterdam"

};

//Act

var query = new Dictionary<String, StringValues>();

var body = JsonConvert.SerializeObject(windSpeedRequest, Formatting.Indented);

var result = ConvertWindSpeedToBeaufort.Run(HttpRequestSetup(query, body), log);

var resultObject = (OkObjectResult)result;

//Assert

var resultResponse = new WindSpeedData

{

WindSpeed = 19.0,

Beaufort = 8,

Location = "Amsterdam"

};

var resultBody = JsonConvert.SerializeObject(resultResponse, Formatting.Indented);

Assert.AreEqual(resultBody, resultObject.Value);

}

[TestMethod]

public void CanConvertWindSpeedToBeaufort9()

{

//Arrange

var windSpeedRequest = new WindSpeedData

{

WindSpeed = 23.0,

Beaufort = 0,

Location = "Amsterdam"

};

//Act

var query = new Dictionary<String, StringValues>();

var body = JsonConvert.SerializeObject(windSpeedRequest, Formatting.Indented);

var result = ConvertWindSpeedToBeaufort.Run(HttpRequestSetup(query, body), log);

var resultObject = (OkObjectResult)result;

//Assert

var resultResponse = new WindSpeedData

{

WindSpeed = 23.0,

Beaufort = 9,

Location = "Amsterdam"

};

var resultBody = JsonConvert.SerializeObject(resultResponse, Formatting.Indented);

Assert.AreEqual(resultBody, resultObject.Value);

}

[TestMethod]

public void CanConvertWindSpeedToBeaufort10()

{

//Arrange

var windSpeedRequest = new WindSpeedData

{

WindSpeed = 28.0,

Beaufort = 0,

Location = "Amsterdam"

};

//Act

var query = new Dictionary<String, StringValues>();

var body = JsonConvert.SerializeObject(windSpeedRequest, Formatting.Indented);

var result = ConvertWindSpeedToBeaufort.Run(HttpRequestSetup(query, body), log);

var resultObject = (OkObjectResult)result;

//Assert

var resultResponse = new WindSpeedData

{

WindSpeed = 28.0,

Beaufort = 10,

Location = "Amsterdam"

};

var resultBody = JsonConvert.SerializeObject(resultResponse, Formatting.Indented);

Assert.AreEqual(resultBody, resultObject.Value);

}

[TestMethod]

public void CanConvertWindSpeedToBeaufort11()

{

//Arrange

var windSpeedRequest = new WindSpeedData

{

WindSpeed = 30.0,

Beaufort = 0,

Location = "Amsterdam"

};

//Act

var query = new Dictionary<String, StringValues>();

var body = JsonConvert.SerializeObject(windSpeedRequest, Formatting.Indented);

var result = ConvertWindSpeedToBeaufort.Run(HttpRequestSetup(query, body), log);

var resultObject = (OkObjectResult)result;

//Assert

var resultResponse = new WindSpeedData

{

WindSpeed = 30.0,

Beaufort = 11,

Location = "Amsterdam"

};

var resultBody = JsonConvert.SerializeObject(resultResponse, Formatting.Indented);

Assert.AreEqual(resultBody, resultObject.Value);

}

[TestMethod]

public void CanConvertHighWindSpeed()

{

//Arrange

var windSpeedRequest = new WindSpeedData

{

WindSpeed = 75.8,

Beaufort = 0,

Location = "Amsterdam"

};

//Act

var query = new Dictionary<String, StringValues>();

var body = JsonConvert.SerializeObject(windSpeedRequest, Formatting.Indented);

var result = ConvertWindSpeedToBeaufort.Run(HttpRequestSetup(query, body), log);

var resultObject = (OkObjectResult)result;

//Assert

var resultResponse = new WindSpeedData

{

WindSpeed = 75.8,

Beaufort = 12,

Location = "Amsterdam"

};

var resultBody = JsonConvert.SerializeObject(resultResponse, Formatting.Indented);

Assert.AreEqual(resultBody, resultObject.Value);

}

[TestMethod]

public void CanConvertNegativeWindSpeed()

{

//Arrange

var windSpeedRequest = new WindSpeedData

{

WindSpeed = -10,

Beaufort = 0,

Location = "Amsterdam"

};

//Act

var query = new Dictionary<String, StringValues>();

var body = JsonConvert.SerializeObject(windSpeedRequest, Formatting.Indented);

var result = ConvertWindSpeedToBeaufort.Run(HttpRequestSetup(query, body), log);

var resultObject = (OkObjectResult)result;

//Assert

var resultResponse = new WindSpeedData

{

WindSpeed = -10,

Beaufort = 0,

Location = "Amsterdam"

};

var resultBody = JsonConvert.SerializeObject(resultResponse, Formatting.Indented);

Assert.AreEqual(resultBody, resultObject.Value);

}

[TestMethod]

public void CanNotConvertWindSpeed()

{

//Arrange

var windSpeedRequest =

"{\r\n \"Location\": \"Amsterdam\",\r\n \"WindSpeed\": ,\r\n \"Beaufort\": 0\r\n}";

//Act

var query = new Dictionary<String, StringValues>();

var body = JsonConvert.SerializeObject(windSpeedRequest, Formatting.Indented);

var result = ConvertWindSpeedToBeaufort.Run(HttpRequestSetup(query, body), log);

var resultObject = (BadRequestObjectResult)result;

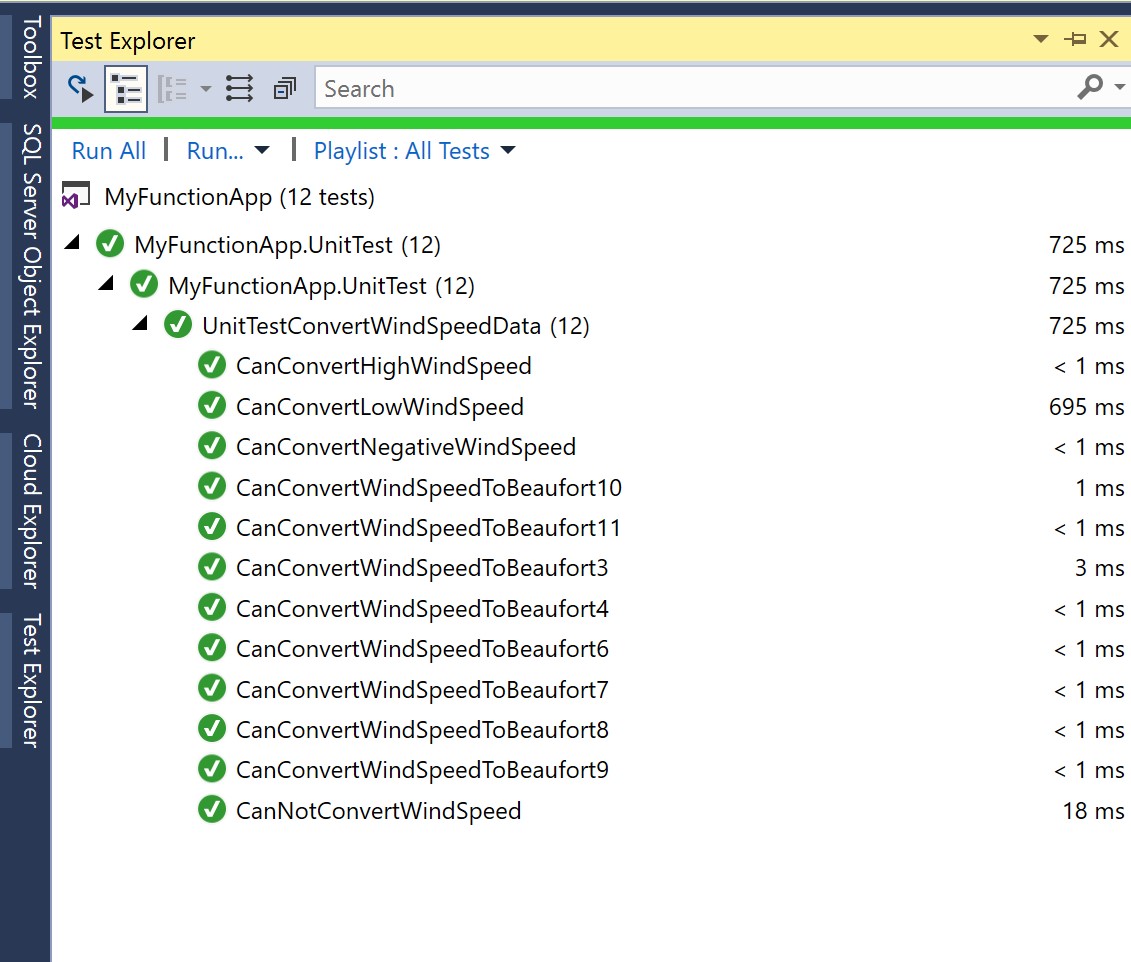
//Assert

Assert.AreEqual("Please pass the correct request body!", resultObject.Value);

}

}

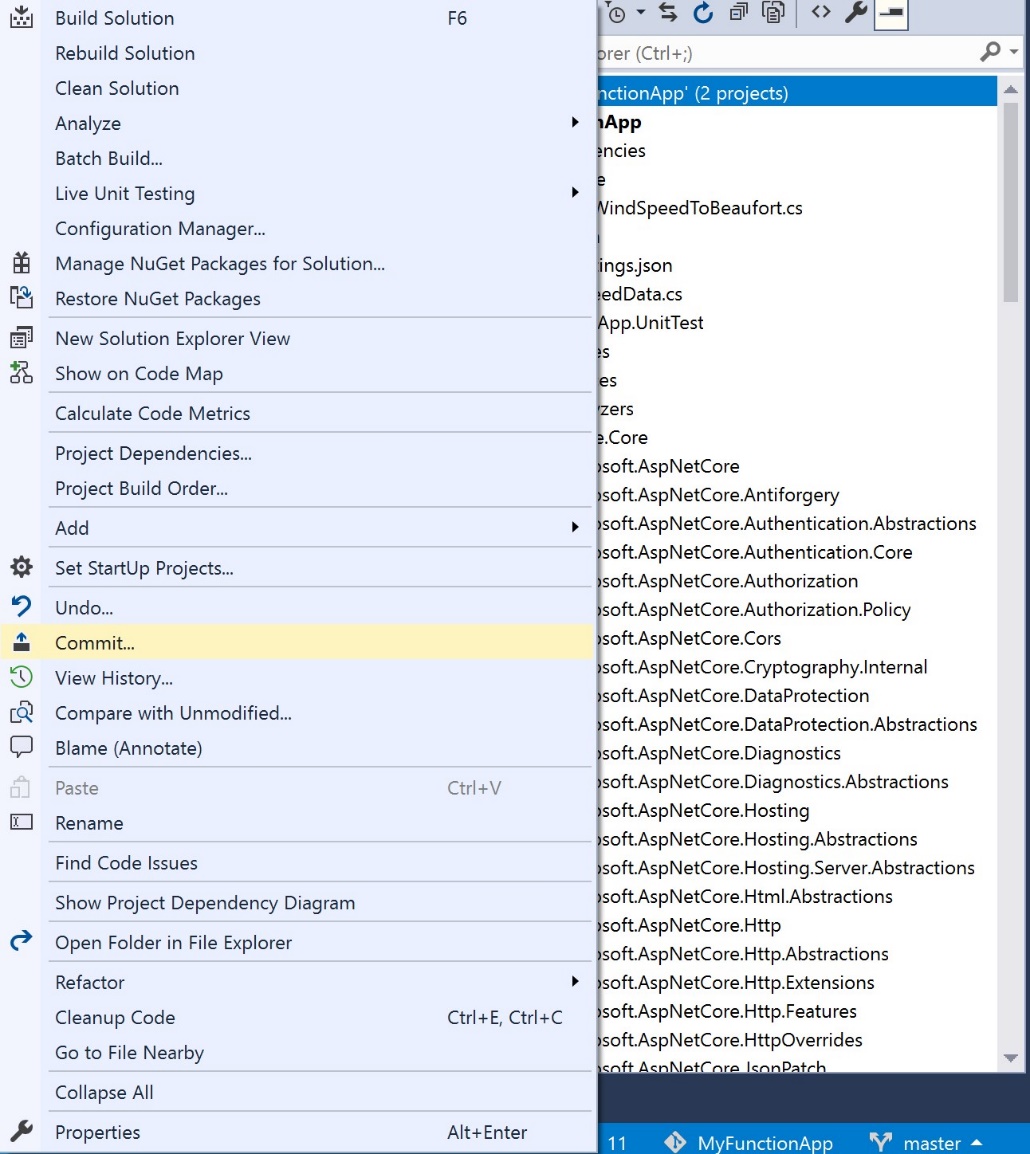
1. Right click Test project and add reference to your Function App project.
2. Build the test project.
3. In the Test menu item choose Run --> All tests.
4. Examine the results in the Test Explorer.



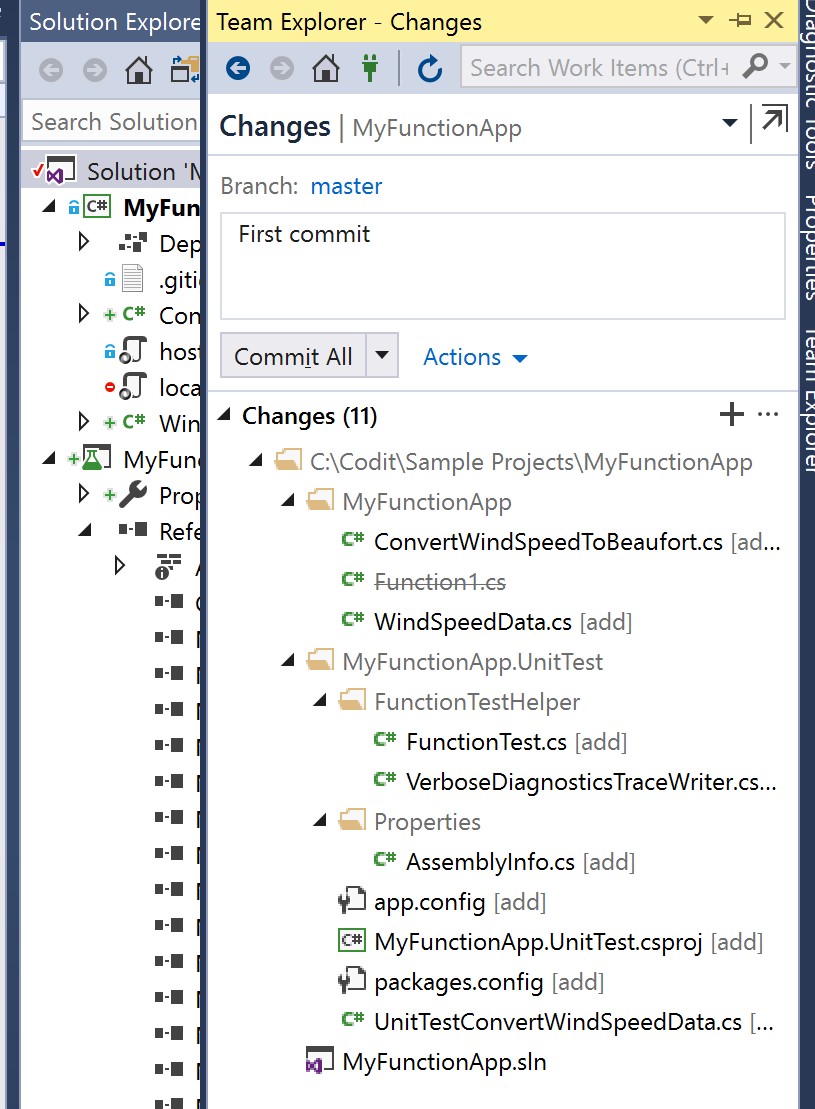
# Step 4 – Push the project to Azure DevOps

In this step you will commit the code to your local repository and subsequently push it to Azure DevOps project.

1. Right click the solution and choose commit.



1. In the pane that will appear, add comment.

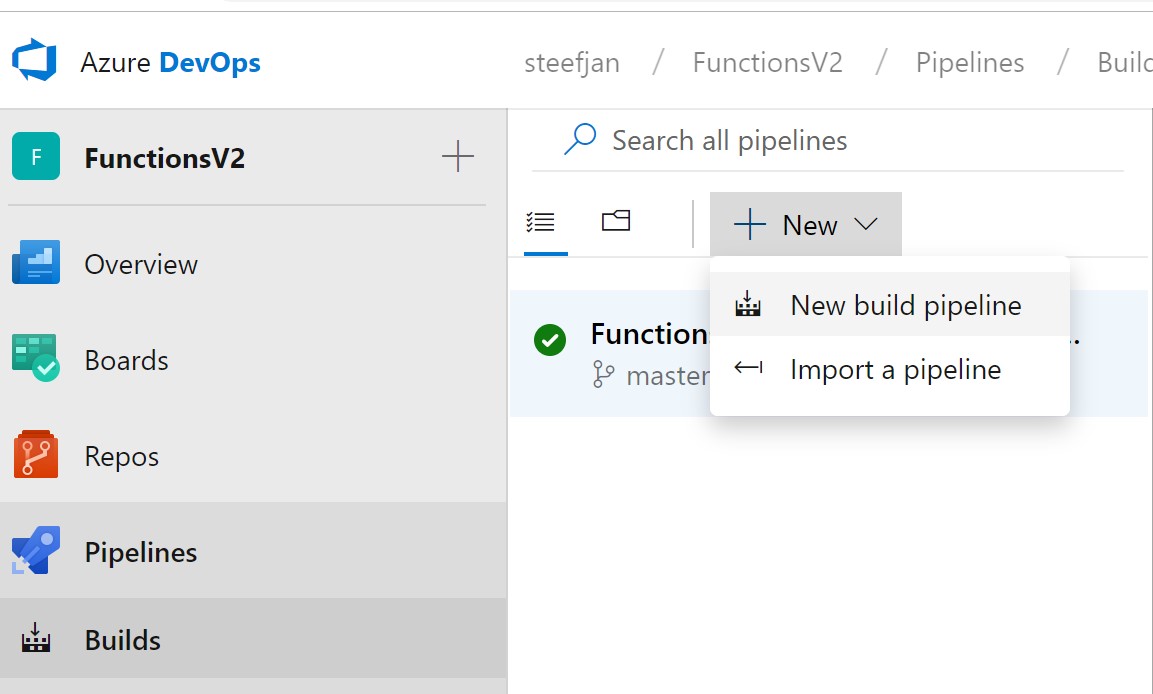


1. Click Commit All.
2. Next click Sync.
3. In outgoing commits click push. The branch (master) will be pushed to the Azure DevOps project.

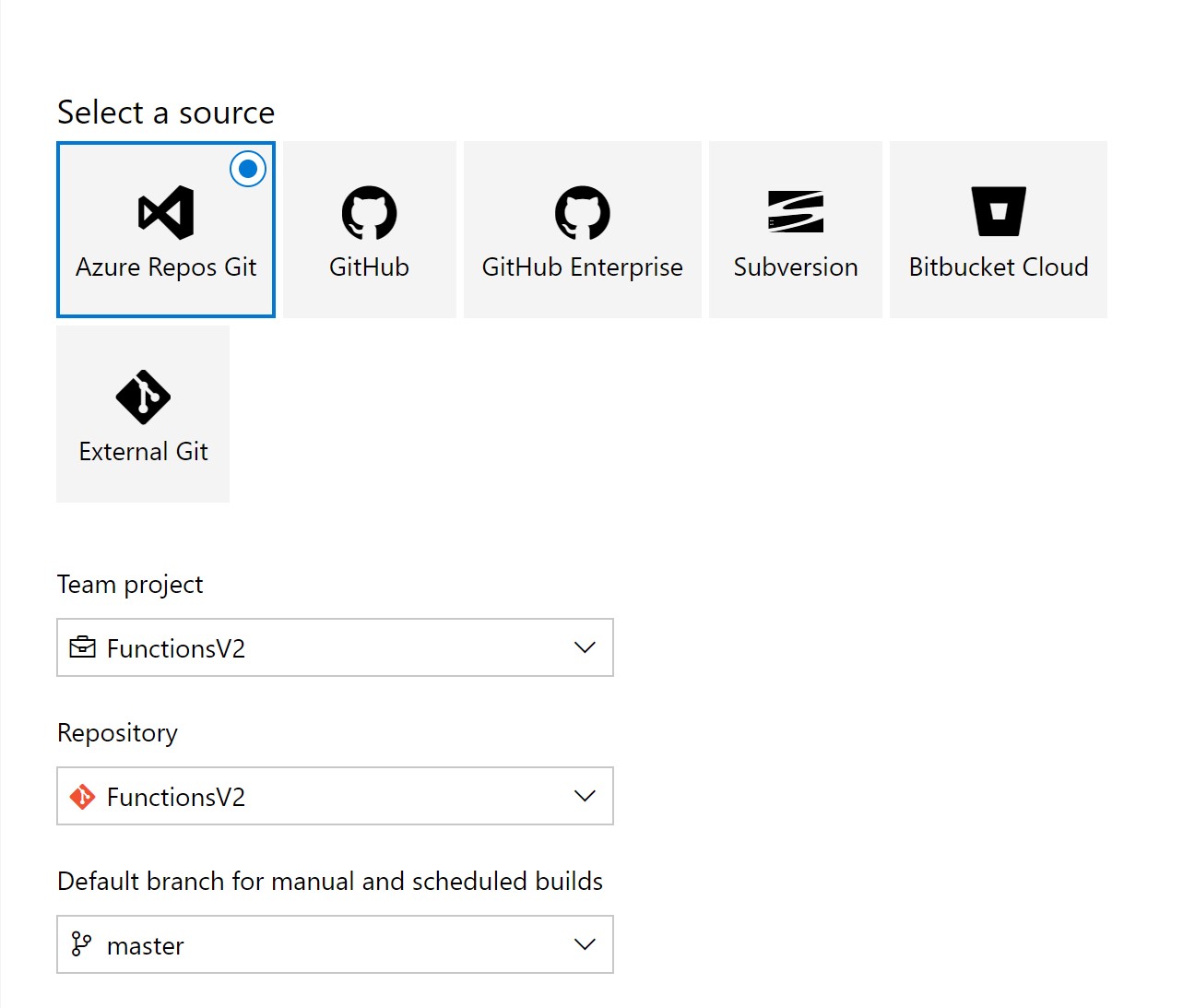
# Step 5 - Create and execute a build template

The code is build and tested locally. Next you pushed it to Azure DevOps. We will now create a build template.

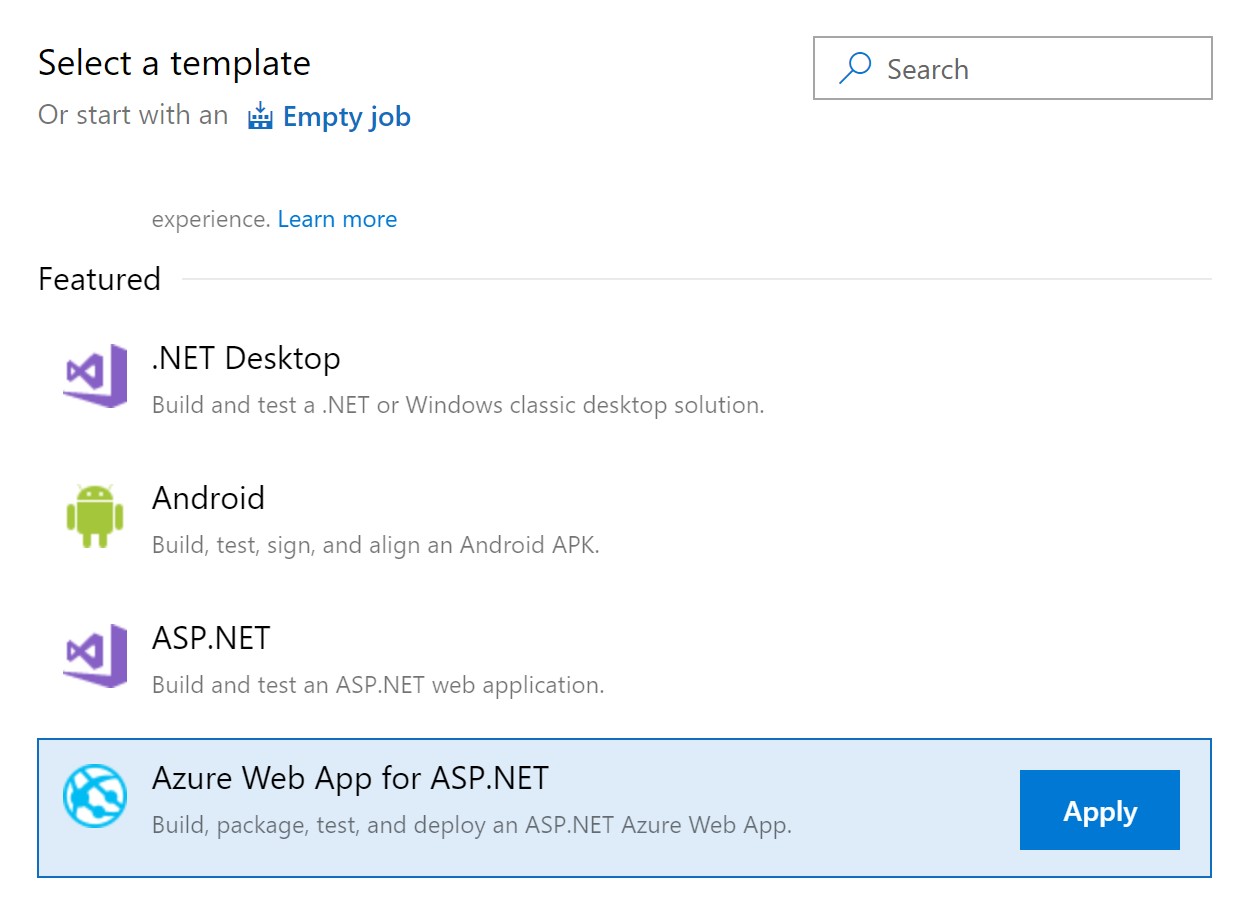
1. Go to Azure DevOps.
2. Click Pipelines in the left hand pane.
3. Click New Build Pipeline.



1. Select you respository



1. Choose Azure Repos Git, you team project, repository, and branch.
2. Choose Azure Web App for ASP.NET.



1. In Azure Subscription click Manage
2. Now you will have to create a managed connection i.e. service principle. You can do this from DevOps portal.
3. This connection is necessary to have appropiate right for the build template.
4. In the template set the connection to the correct Azure Subscription and choose the correct app service i.e. a function app you create earlier (lab 6).
5. In the Build solution step – MS Build Arguments paste the following:

/p:DeployOnBuild=true /p:WebPublishMethod=Package /p:PackageAsSingleFile=true

/p:SkipInvalidConfigurations=true

/p:DesktopBuildPackageLocation="$(build.artifactstagingdirectory)\WebApp.zip"

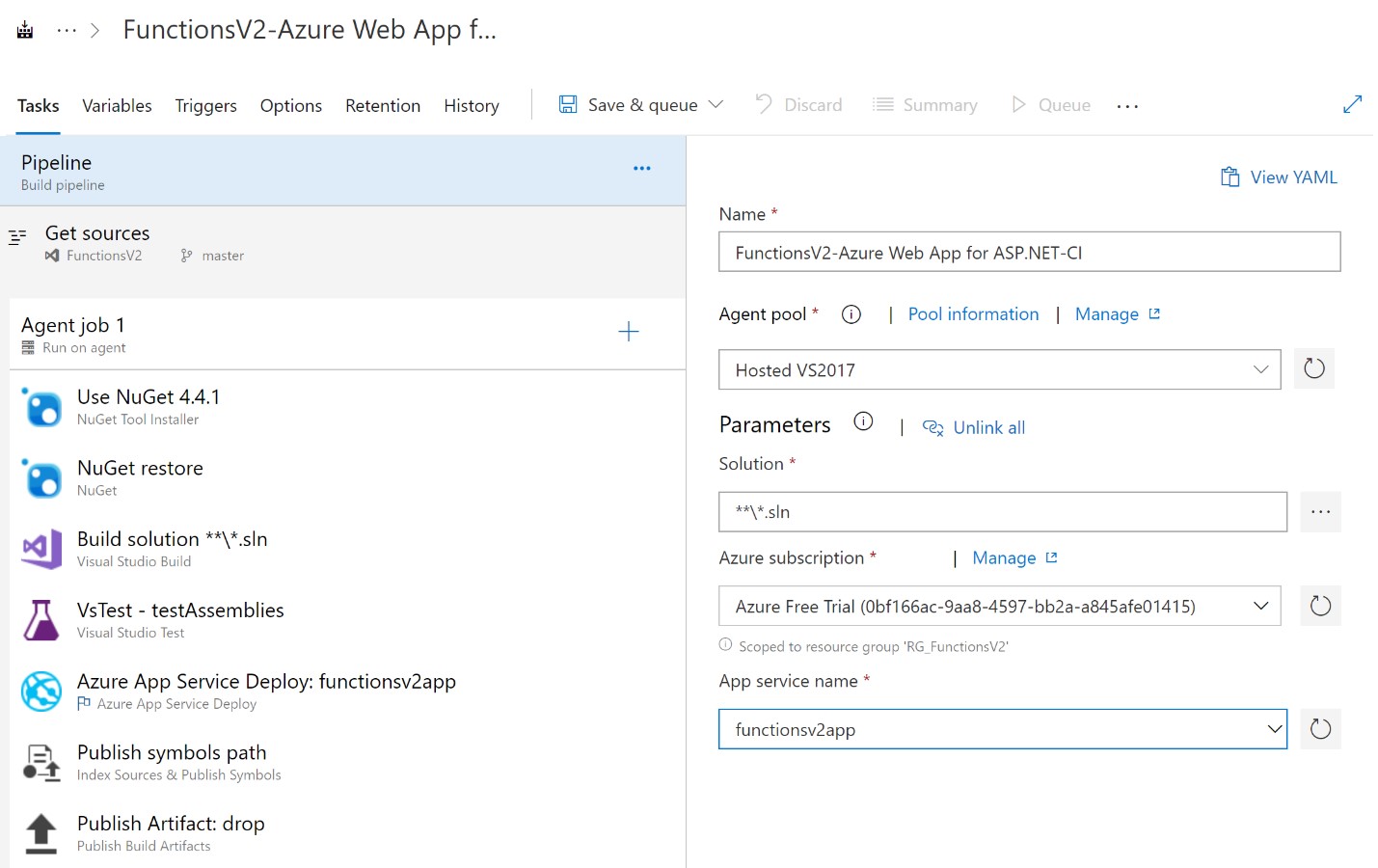
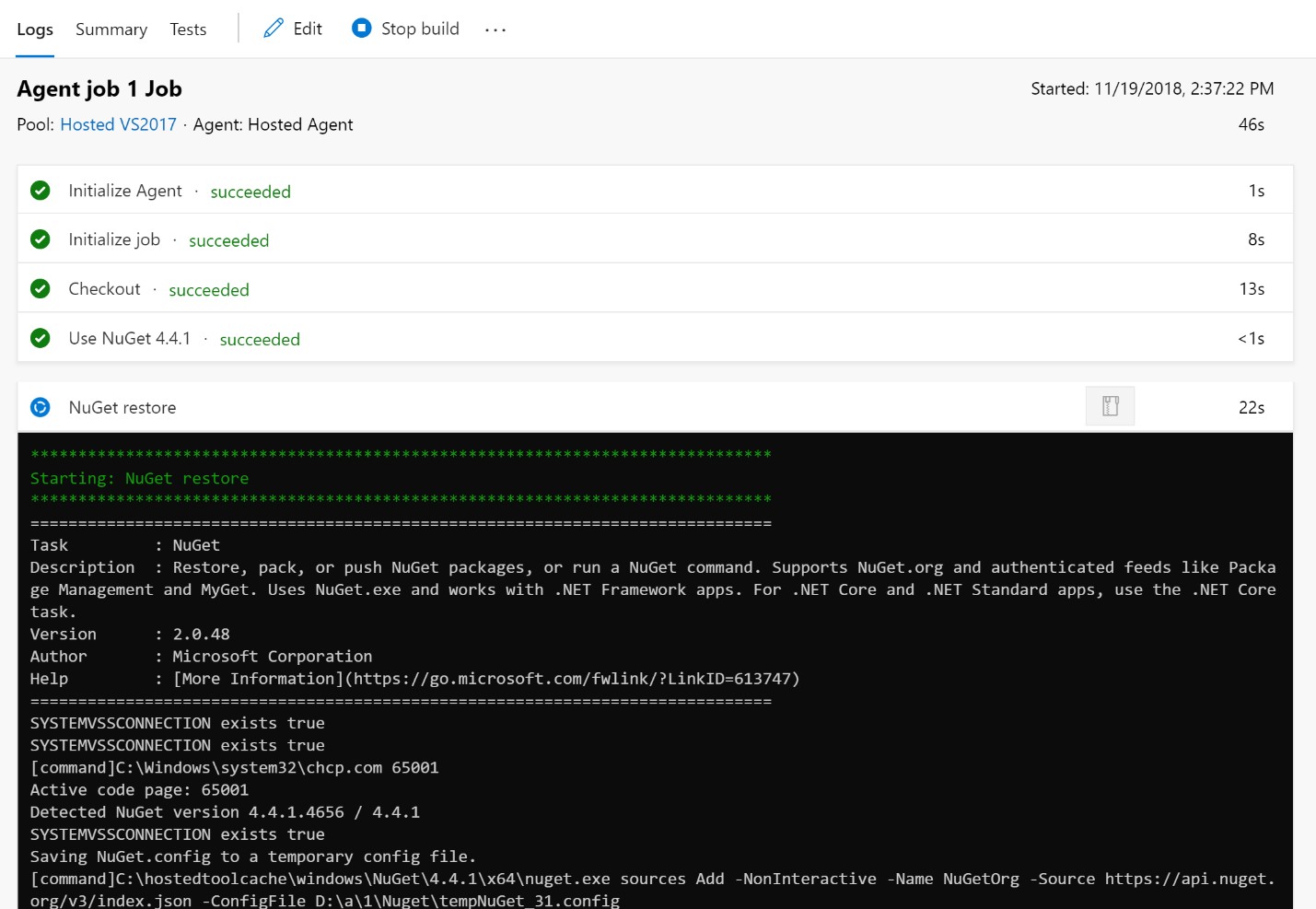
/p:DeployIisAppPath="Default Web Site"

1. In the Test Assemblies – Test files paste the following:

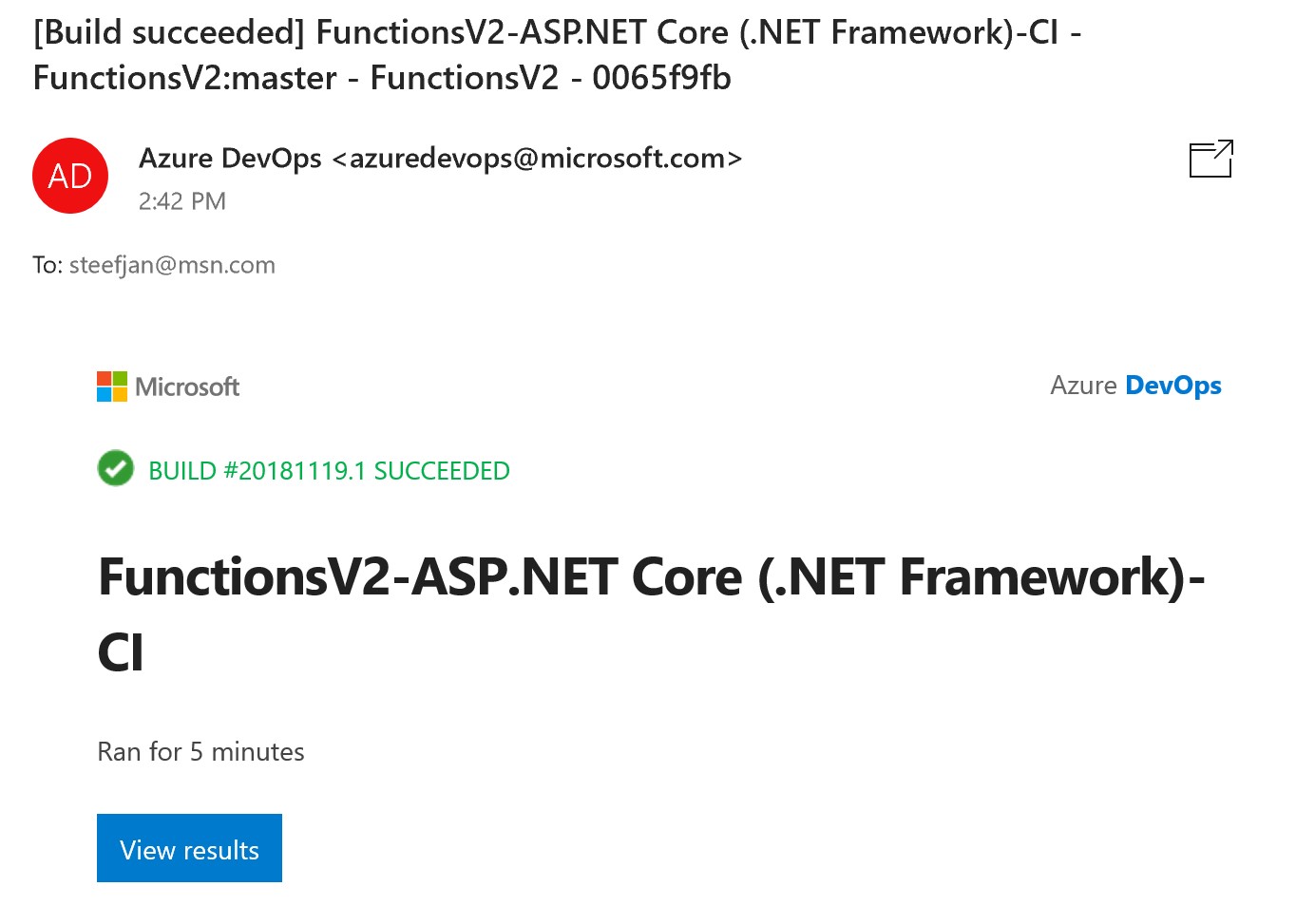
\*\*\$(BuildConfiguration)\\*test\*.dll

!\*\*\obj\\*\*

1. Click Save and Queue. The build template will now execute.
2. Click the build tlink hat will appear.
3. A new pane will appear where you can follow the build process.



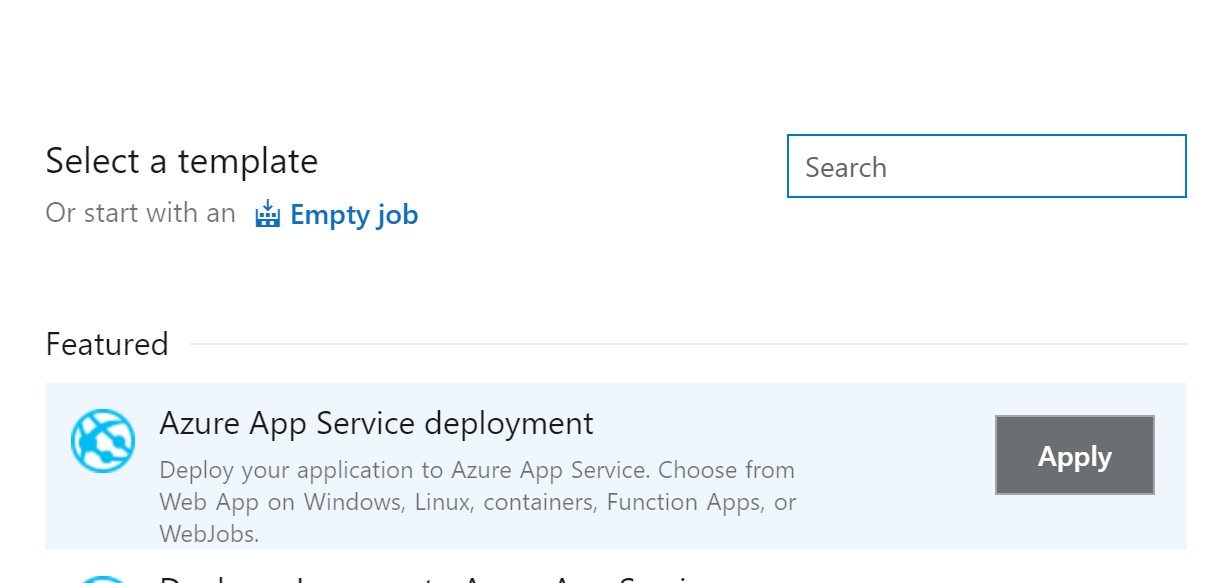
1. Once the build is complete you will get an email.



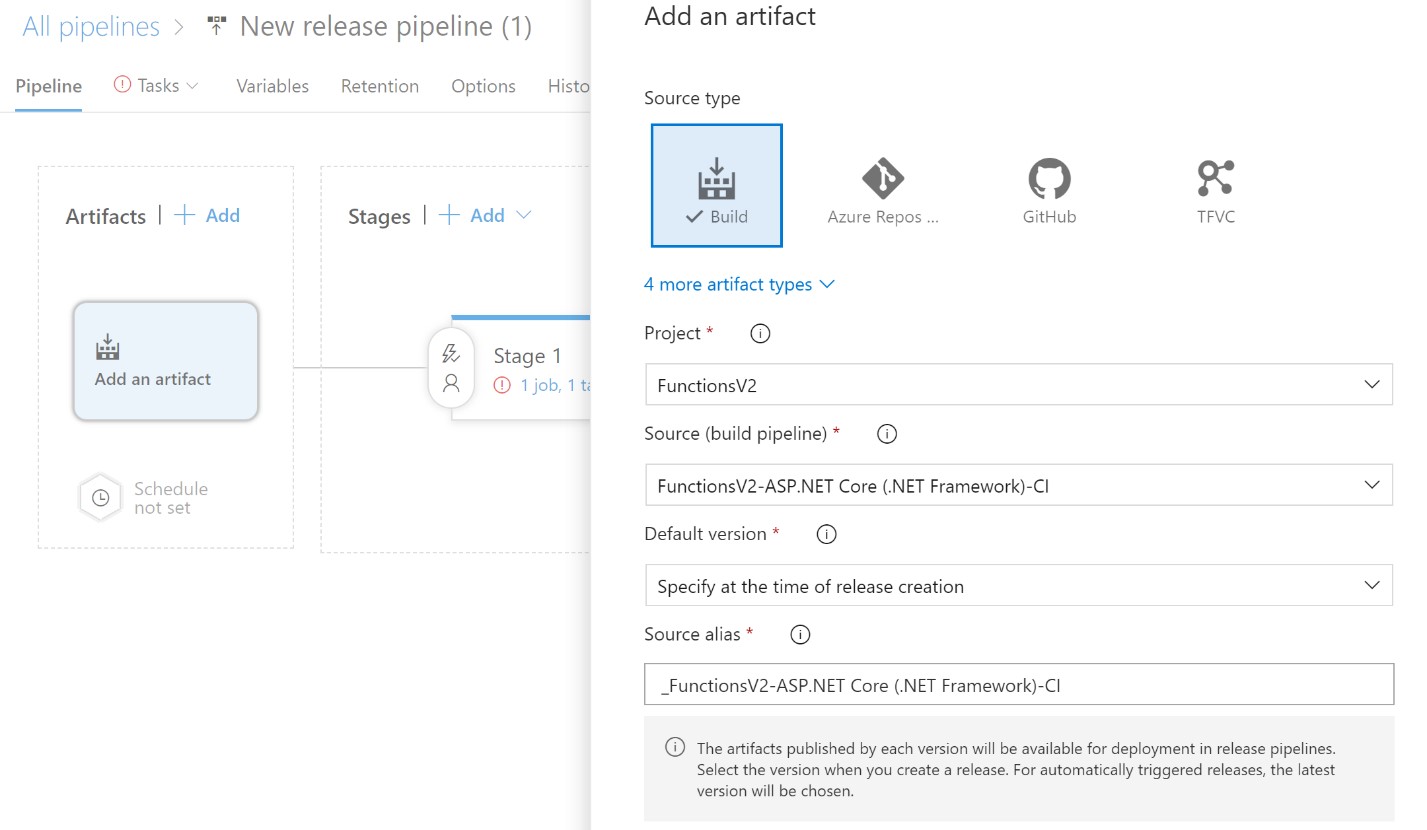
# Step 6 - Create and execute a release template

In the final step of this lab we will build a release template based on our build (step 5) and execute it.

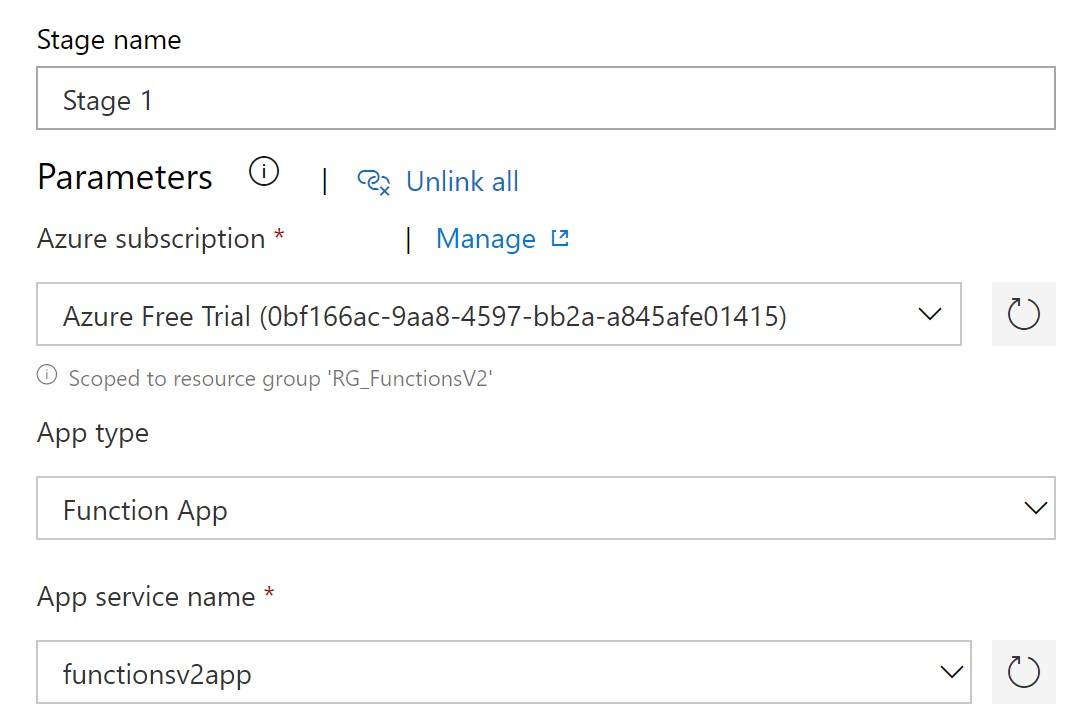
1. In the left hand pane select release pipeline.
2. Click New release pipeline.
3. Choose Azure App Service deployment.



1. Click **Apply**.
2. Click Add artifact, which is the build from step 5.



1. Click **Add**.
2. Go to stage 1 and select Deploy Azure App Service Task.
3. Set the parameters to correct Azure Subscription.
4. Set the correct App Type and App Service Name.



1. Create a release.
2. Observe the release progress.

