Lab 3 – Azure Functions and DevOps

In this lab, you will familiarize yourself with Azure DevOps Build and Release Templates.

Objective

In this lab, you will build an Azure Function locally, push the code to a respository, create and execute a build template, and do a release.

Prerequisites

- Azure Subscription
- Azure DevOps

Steps

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**.

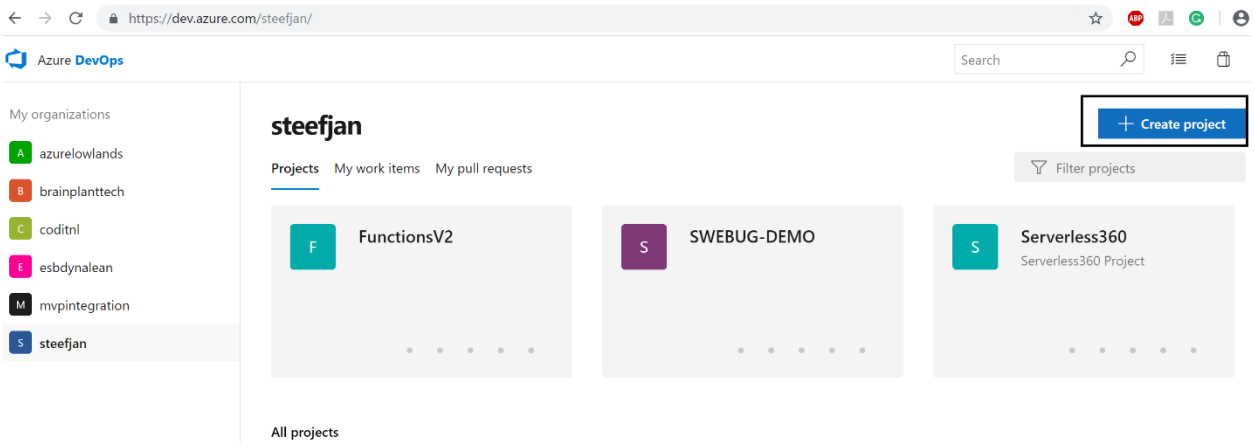
Created by Steef-Jan Wiggers, Microsoft Azure MVP, Codit Domain Lead Azure

Contact steef-jan.wiggers@codit.eu or twitter @steefJan

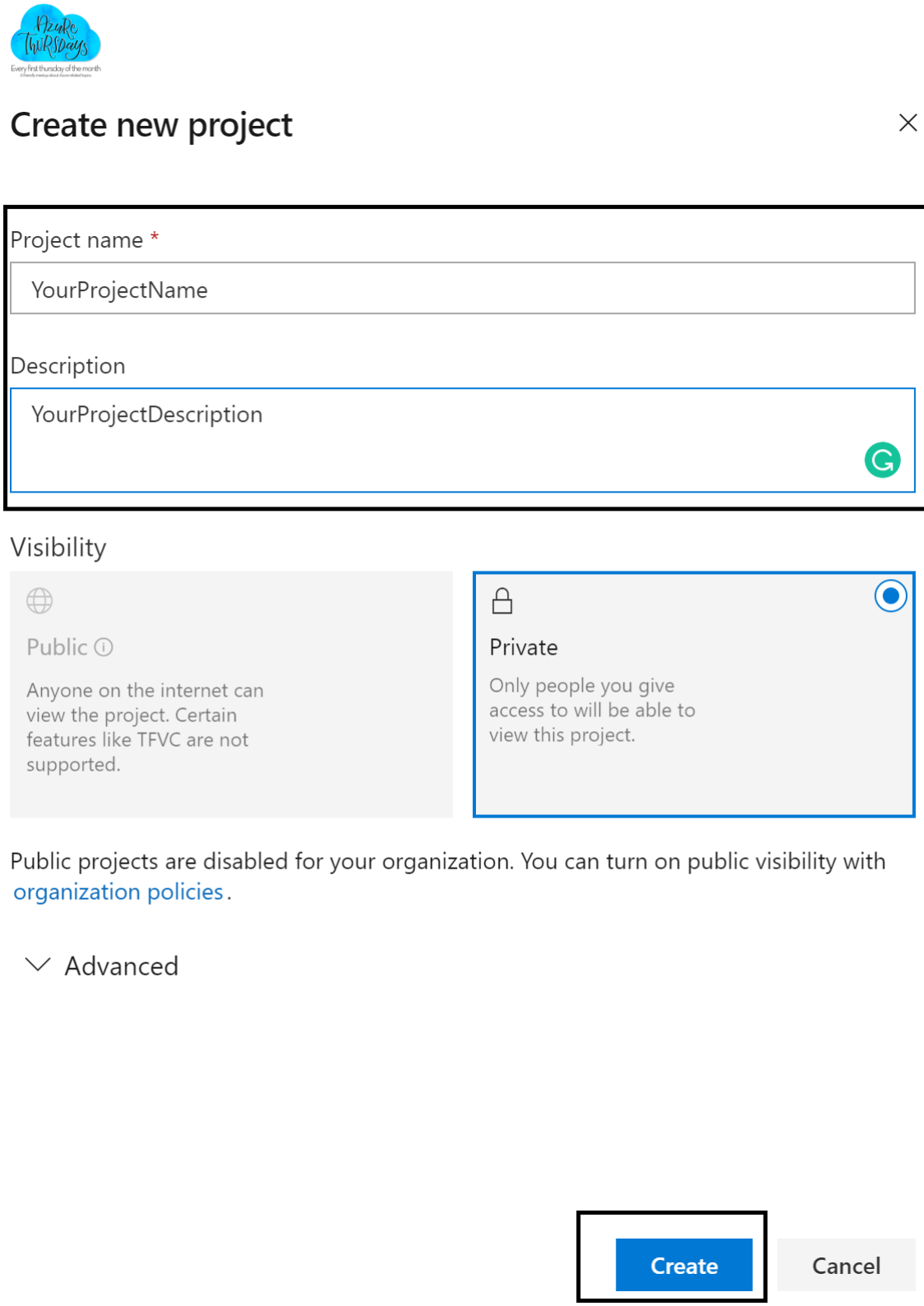
Step 1 - Create an Azure DevOps Project

The very first step in this lab is

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



4. A new pane will appear.
5. Fill a descriptive name and description and click **Create**.

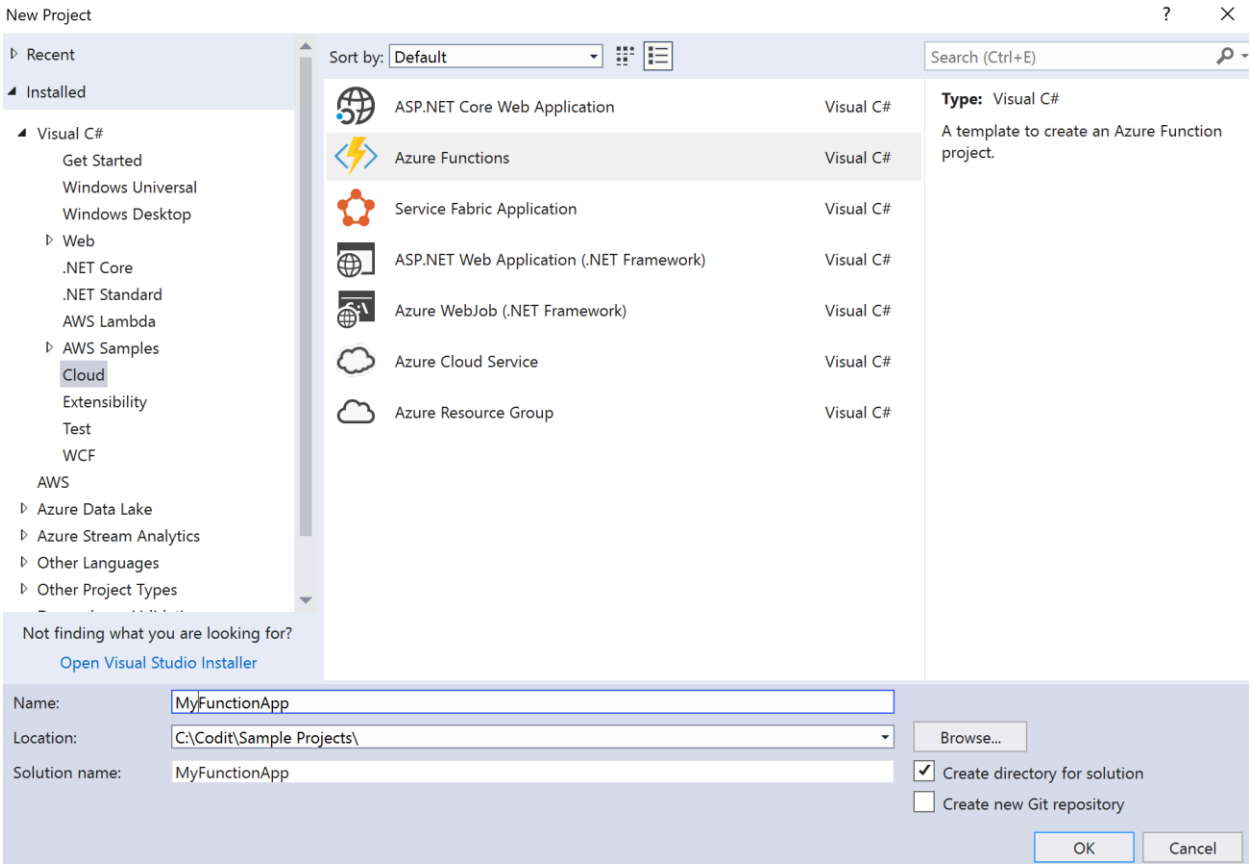


6. You have now a project in Azure DevOps.

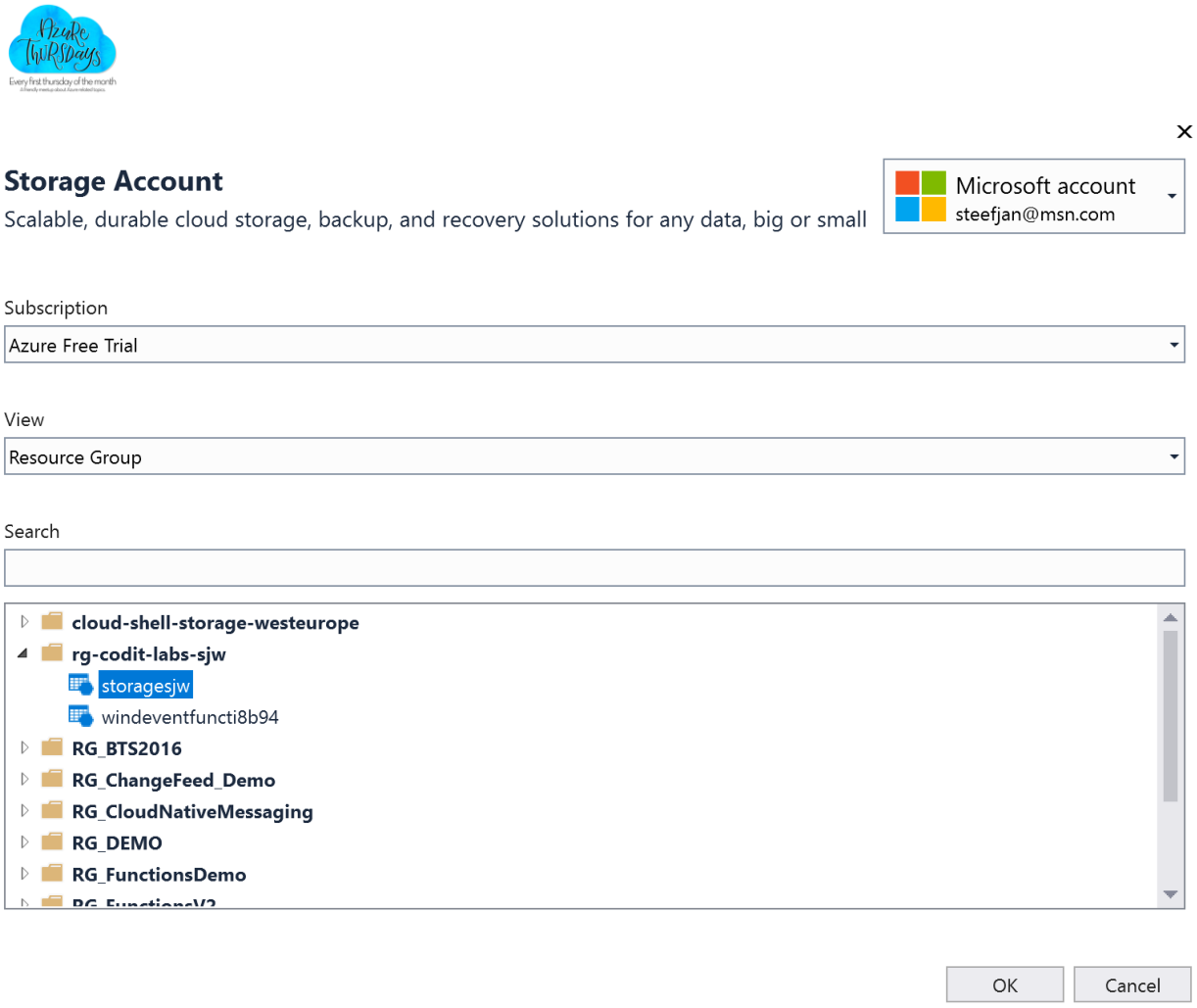
Step 2 - Create an Azure Function and test locally

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

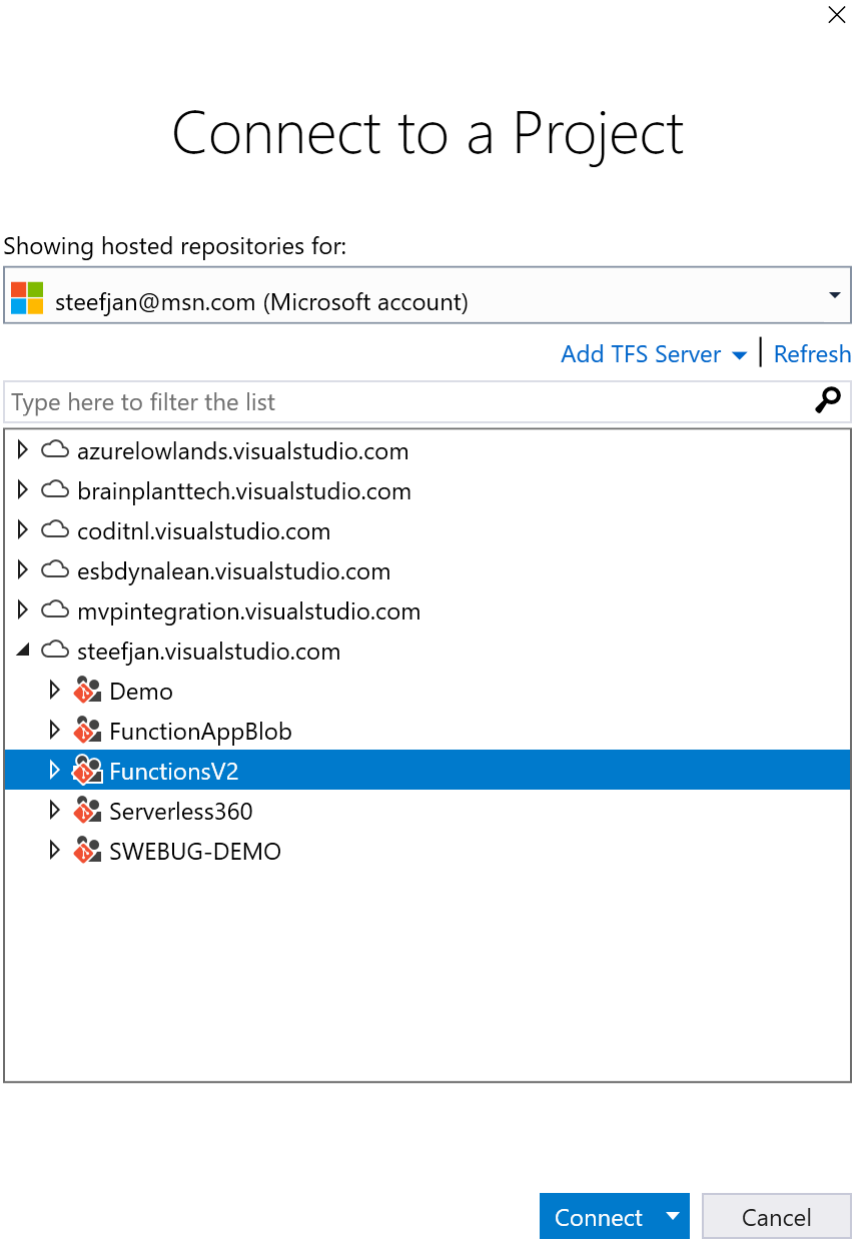
1. Open Visual Studio 2017.
2. Select **File** --> **New** --> **Project**
3. In the New Project Pane, select **Cloud**.
4. Subsequently, choose **Azure Functions**.
5. Provide a name, choose the correct path, and solution name.



6. Click **Ok**.
7. Choose Azure Functions V2.
8. For Storage Account choose ***Browse***.
9. Pick a storage account from earlier lab or create a new one.



10. Click **Ok**.
11. Choose the **HTTP Trigger Template** and click **Ok**.
12. Click **Team Explorer** on the right side of Visual Studio.
13. Click manage connections.
14. Click Connect to a project.
15. It will show the repositories belonging to your account.
16. Select the project you created in step 1 and click **Connect**.



17. Right click solution and choose add to source control.
18. Rename function1.cs to ConvertWindSpeedToBeaufort.cs.
19. 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;
```

20. 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!");
```

```
        }
    }
```

21. 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; }

    }
```

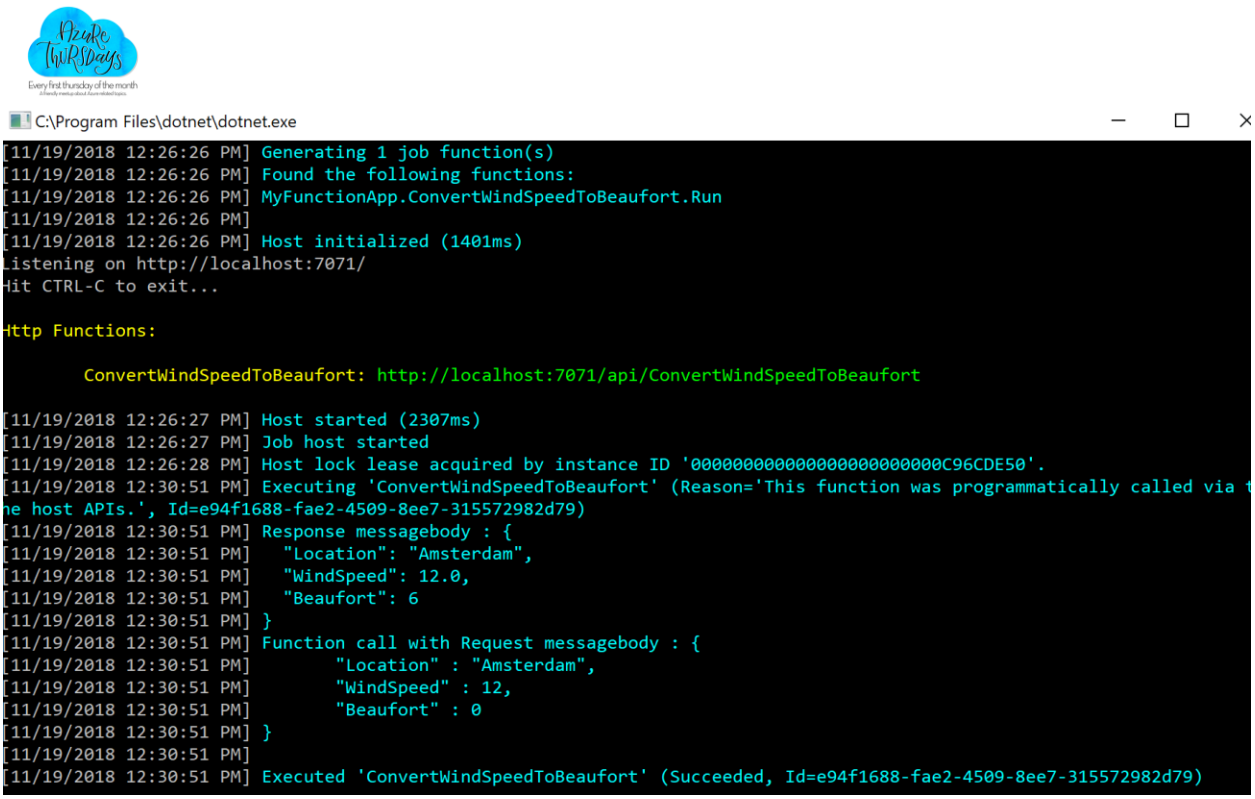
23. Build the project.
24. Run the project.
25. Copy the endpoint http://localhost:7071/api/ConvertWindSpeedToBeaufort
26. Open Postman, and create a POST to the URL
27. Use the following payload:

```
{
        "Location" : "Amsterdam",
        "WindSpeed" : 12,
        "Beaufort" : 0
}
```

28. Click Send.
29. You will get the following response:

```
{
  "Location": "Amsterdam",
  "WindSpeed": 12.0,
  "Beaufort": 6
}
```

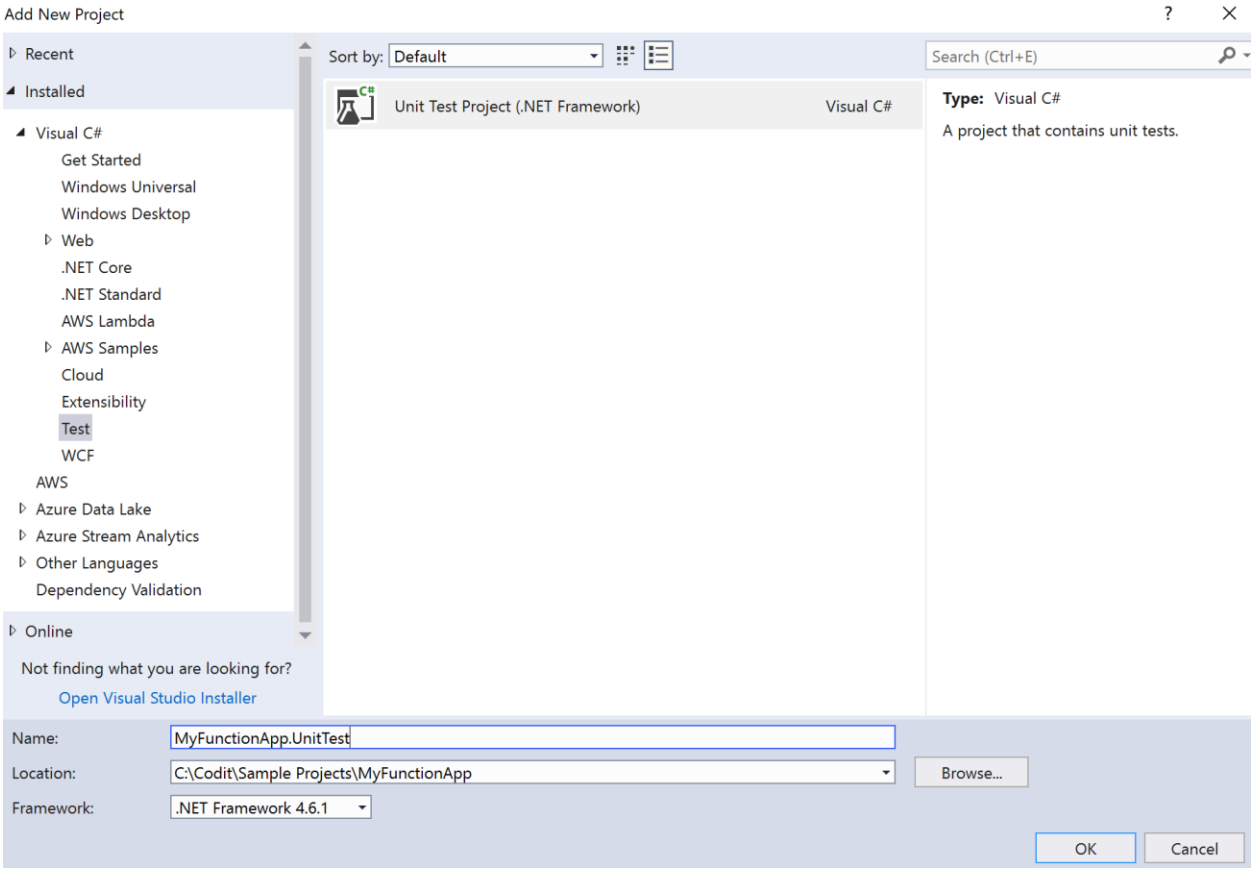
30. Observe the console window that fired up after running the function.


C:\Program Files\dotnet\dotnet.exe
[11/19/2018 12:26:26 PM] Generating 1 job function(s)
[11/19/2018 12:26:26 PM] Found the following functions:
[11/19/2018 12:26:26 PM] MyFunctionApp.ConvertWindSpeedToBeaufort.Run
[11/19/2018 12:26:26 PM]
[11/19/2018 12:26:26 PM] Host initialized (1401ms)
Listening on http://localhost:7071/
Hit CTRL-C to exit...
Http Functions:
ConvertWindSpeedToBeaufort: http://localhost:7071/api/ConvertWindSpeedToBeaufort
[11/19/2018 12:26:27 PM] Host started (2307ms)
[11/19/2018 12:26:27 PM] Job host started
[11/19/2018 12:26:28 PM] Host lock lease acquired by instance ID '000000000000000000000000C96CDE50'.
[11/19/2018 12:30:51 PM] Executing 'ConvertWindSpeedToBeaufort' (Reason='This function was programmatically called via t
he host APIs.', Id=e94f1688-fae2-4509-8ee7-315572982d79)
[11/19/2018 12:30:51 PM] Response messagebody : {
[11/19/2018 12:30:51 PM] "Location": "Amsterdam",
[11/19/2018 12:30:51 PM] "WindSpeed": 12.0,
[11/19/2018 12:30:51 PM] "Beaufort": 6
[11/19/2018 12:30:51 PM] }
[11/19/2018 12:30:51 PM] Function call with Request messagebody : {
[11/19/2018 12:30:51 PM] "Location" : "Amsterdam",
[11/19/2018 12:30:51 PM] "WindSpeed" : 12,
[11/19/2018 12:30:51 PM] "Beaufort" : 0
[11/19/2018 12:30:51 PM] }
[11/19/2018 12:30:51 PM]
[11/19/2018 12:30:51 PM] Executed 'ConvertWindSpeedToBeaufort' (Succeeded, Id=e94f1688-fae2-4509-8ee7-315572982d79)

Step 3 - Add Unittests 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.



5. Click **Ok**.
6. Right Click the newly added test project and choose New Folder
7. Name it FunctionTestHelper
8. Rightclick the folder and add New Item.
9. Choose Class and name it FunctionTest.cs.
10. 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;
```

11. 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);
        }
    }
```

12. Rightclick the folder and add New Item.
13. Choose Class and name it FunctionTest.cs.
14. Above the namespace past the following code:

```
using Microsoft.Azure.WebJobs.Host;
```

```
using System.Diagnostics;
```

15. 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);
        }
    }
```

16. Right click test project and choose manage nuget packages.
17. Find Castle.core and install it.
18. Next find Microsoft.AspNetCore and choose version 2.0.1 and install.
19. Next find Microsoft.Azure.WebJobs and choose version 3.0.0-beta5 (note in the search check the **include prereleases**) and install it.
20. Next, find Microsoft.AspNetCore.Mvc and choose version 2.0.2 and install it.
21. Lastly, find Moq and choose version 4.10 and install it.
22. Rename UnitTest1.cs to UnitTestConvertWindSpeedData.cs
23. 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;
```

24. 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);
```

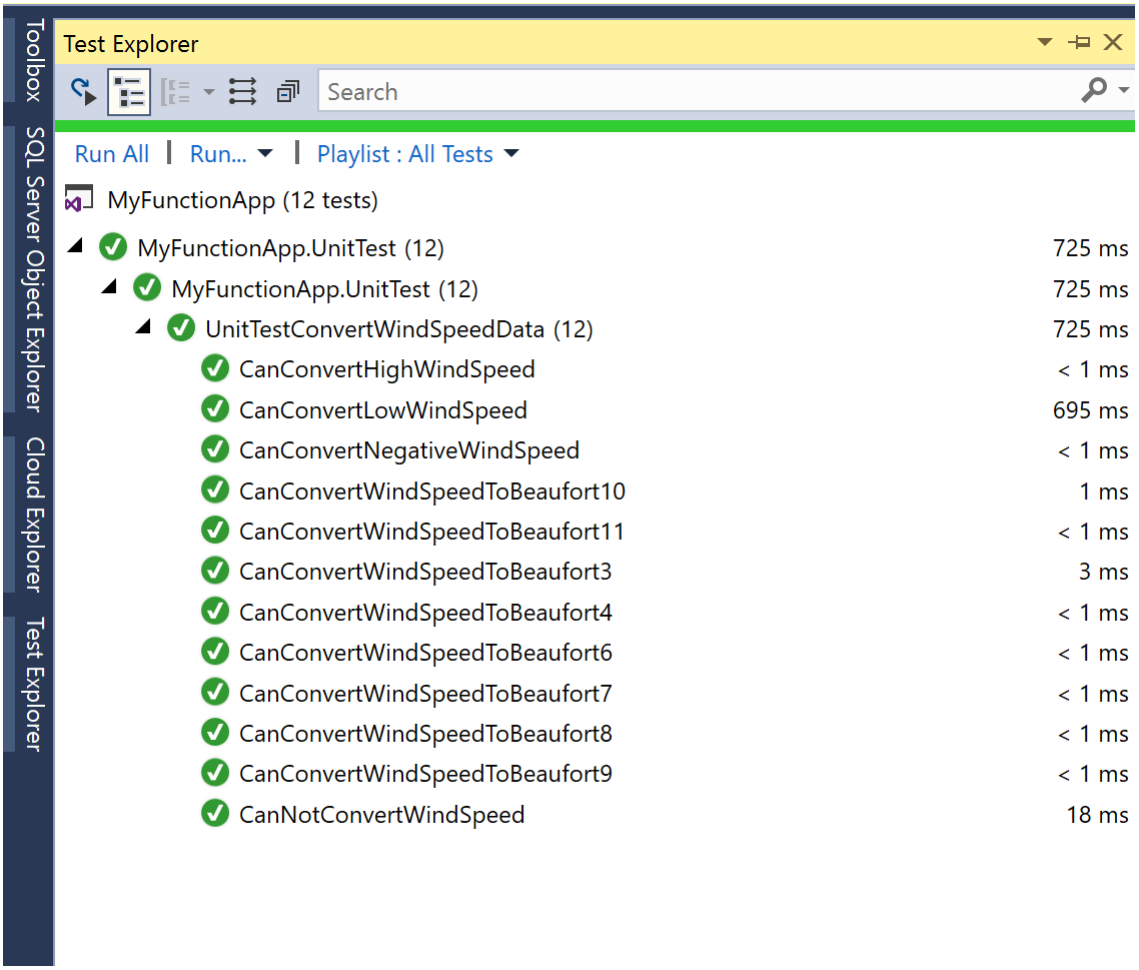
```
    Assert.AreEqual(resultBody, resultObject.Value);
}

[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);
    }
}
```

25. Right click Test project and add reference to your Function App project.
26. Build the test project.
27. In the Test menu item choose Run --> All tests.

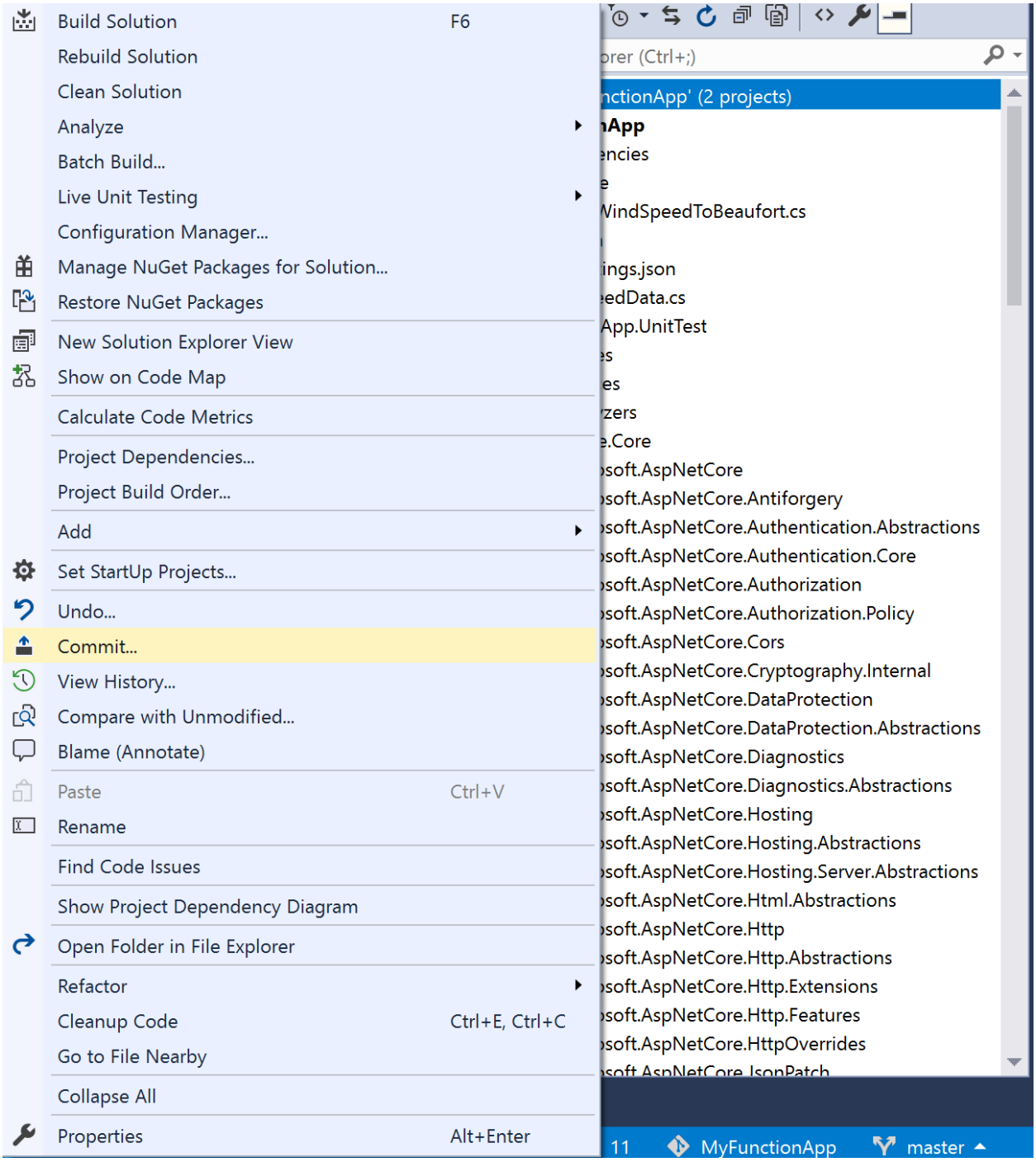
28. Examine the results in the Test Explorer.



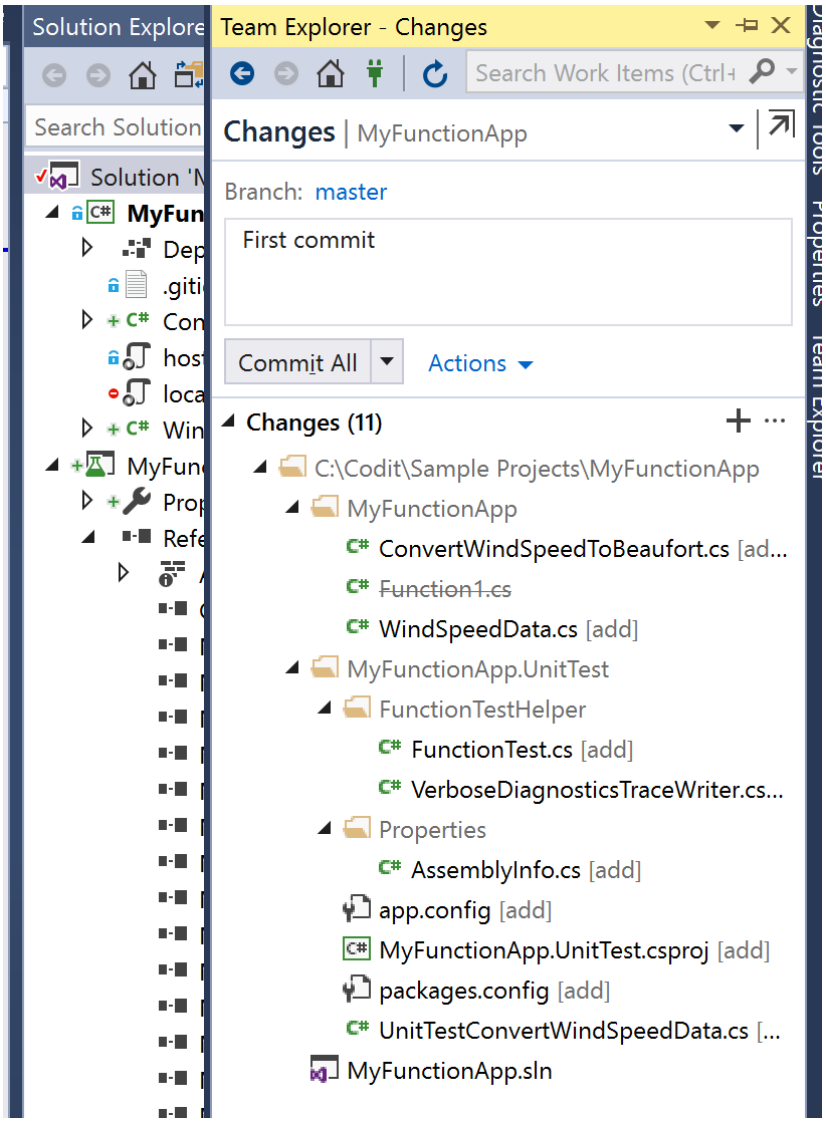
Step 4 – Push the project to Azure DevOps

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.



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

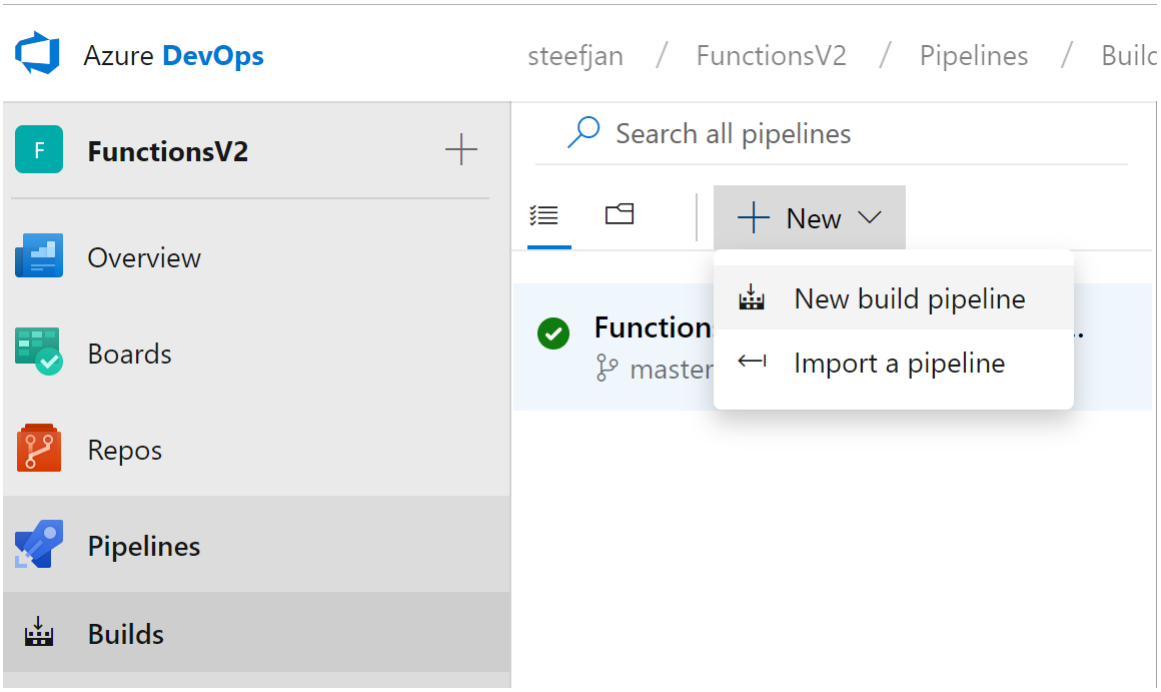


3. Click Commit All.
4. Next click Sync.
5. In outgoing commits click push. The branch (master) will be pushed to the Azure DevOps project.

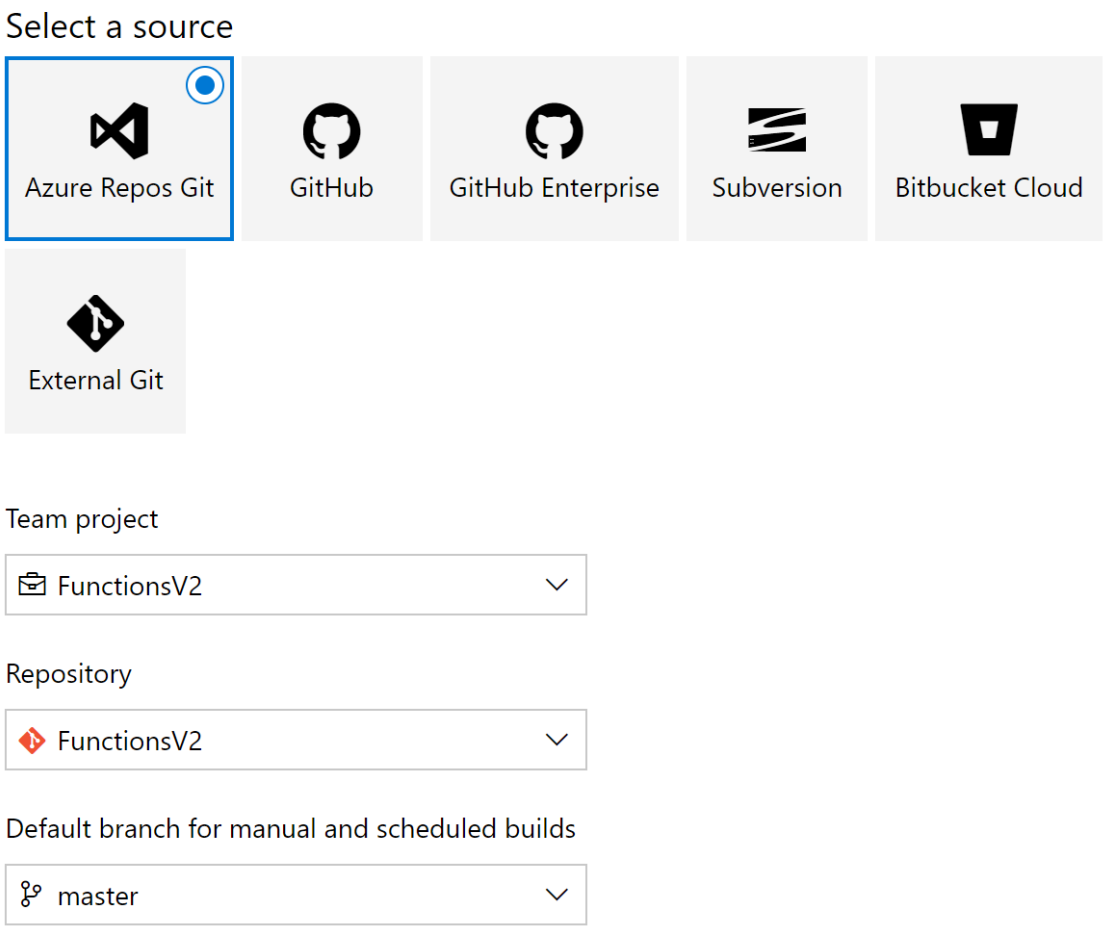
Step 5 - Create and execute a build template

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

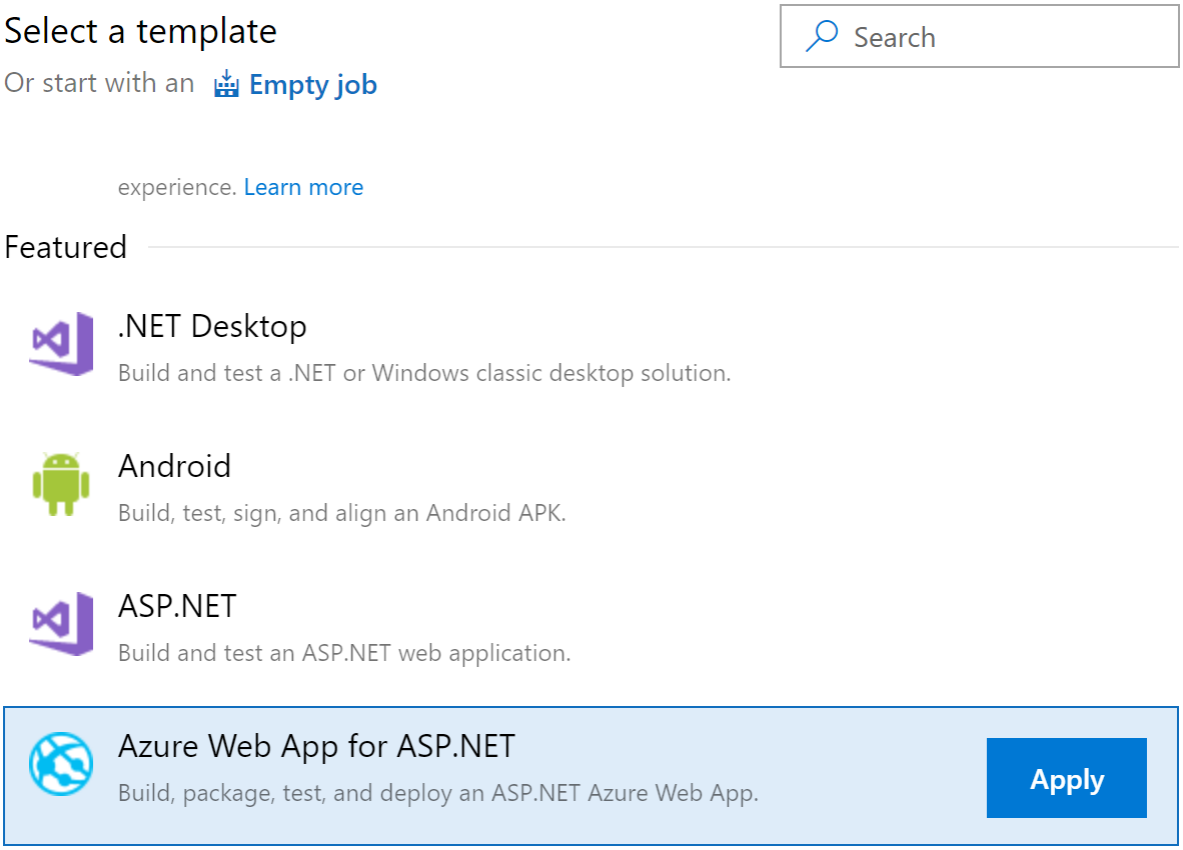
1. Go to Azure DevOps.
2. Click Pipelines in the left hand pane.
3. Click New Build Pipeline.



4. Select you respository



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



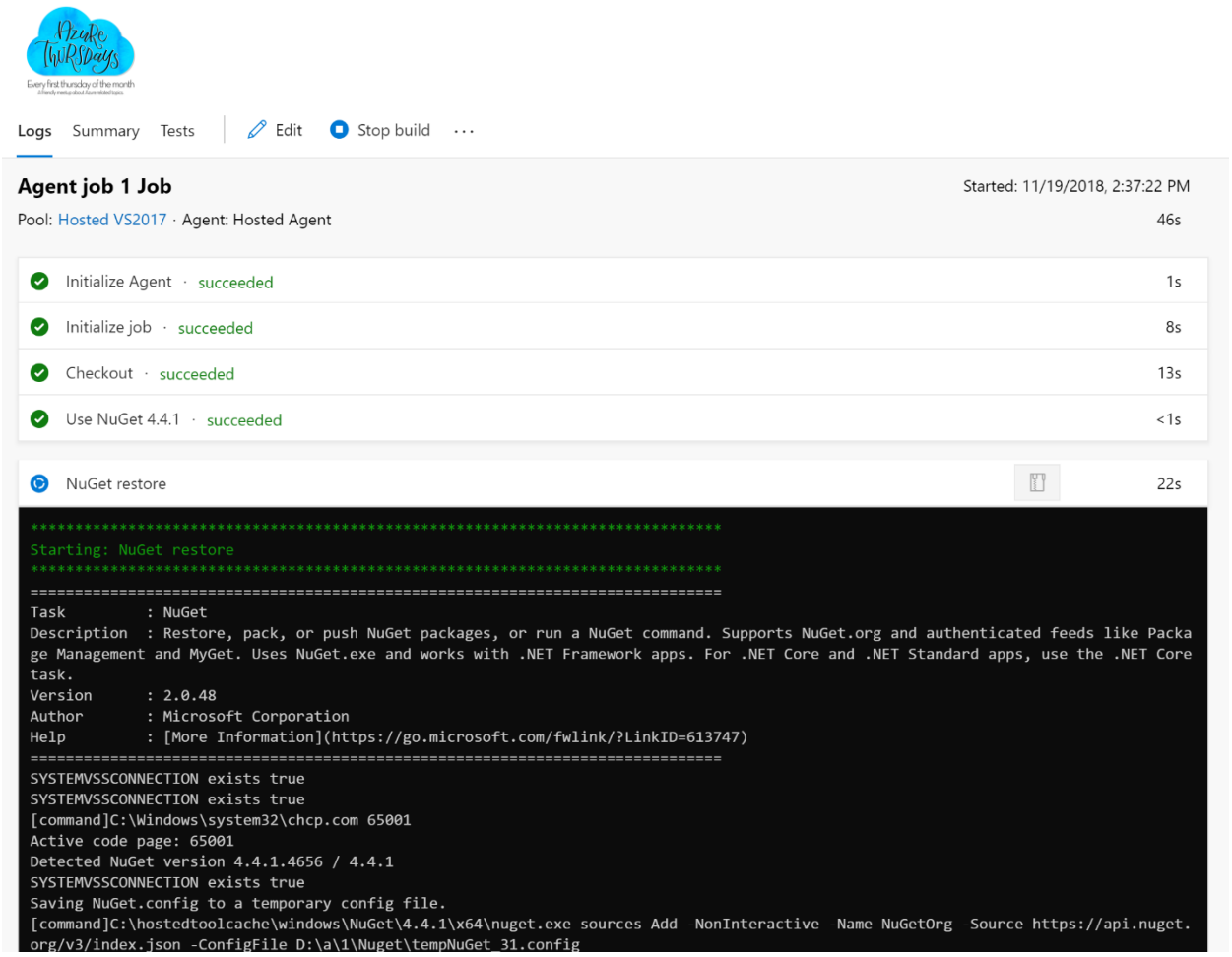
7. In Azure Subscription click Manage
8. Now you will have to create a managed connection i.e. service principle. You can do this from DevOps portal.
9. This connection is necessary to have appropiate right for the build template.
10. In the template set the connection to the correct Azure Subscription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

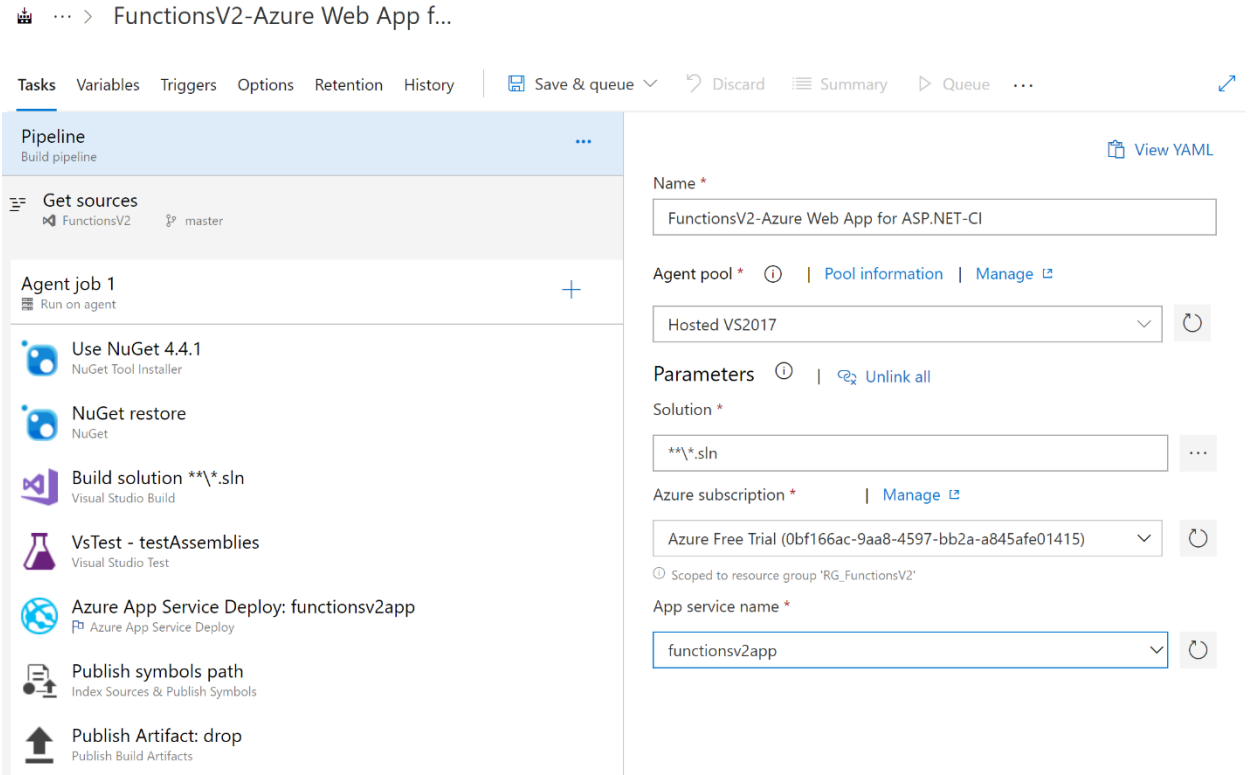
```
/p:DeployOnBuild=true /p:WebPublishMethod=Package /p:PackageAsSingleFile=true
/p:SkipInvalidConfigurations=true
/p:DesktopBuildPackageLocation="$(build.artifactstagingdirectory)\WebApp.zip"
/p:DeployIisAppPath="Default Web Site"
```

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

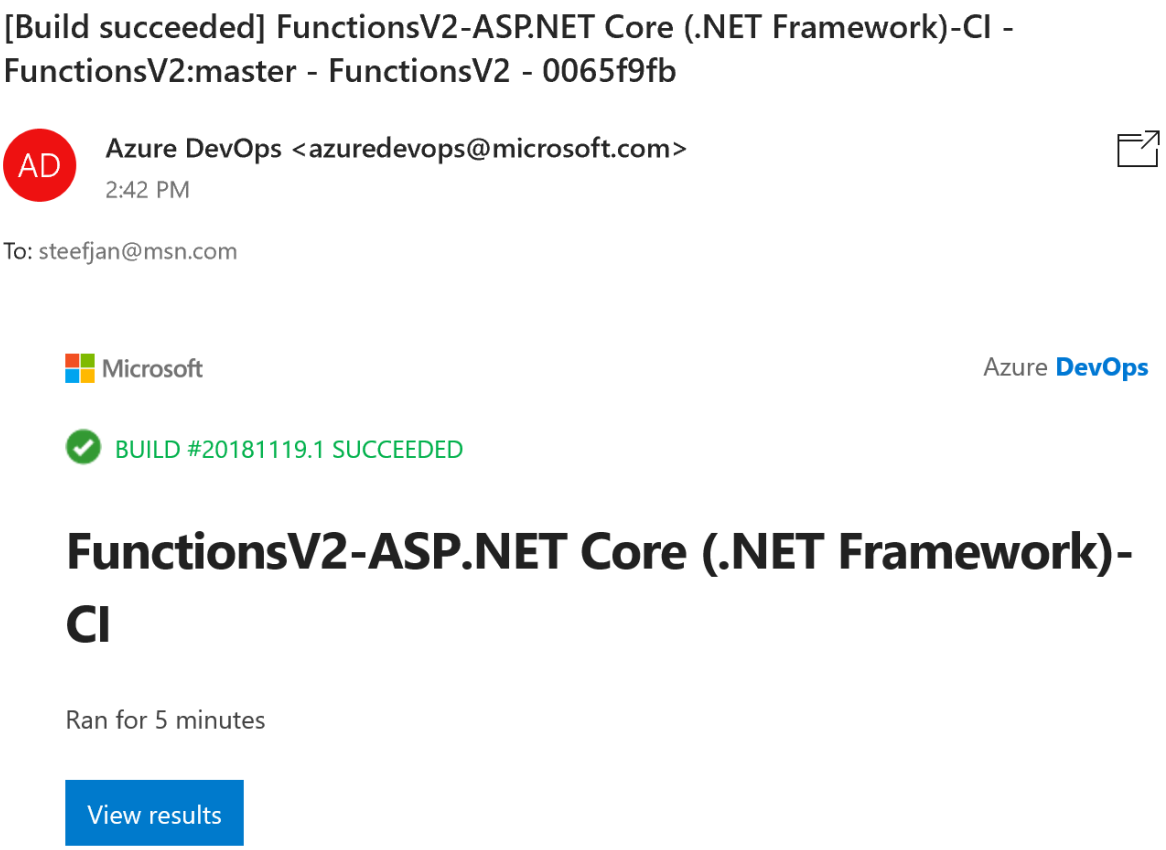
```
**\$(BuildConfiguration)\*test*.dll
!**\obj\**
```

13. Click Save and Queue. The build template will now execute.
14. Click the build tlink hat will appear.
15. A new pane will appear where you can follow the build process.


Logs Summary Tests Edit Stop build ...
Agent job 1 Job
Started: 11/19/2018, 2:37:22 PM
Pool: Hosted VS2017 · Agent: Hosted Agent
46s
Initialize Agent · succeeded 1s
Initialize job · succeeded 8s
Checkout · succeeded 13s
Use NuGet 4.4.1 · succeeded <1s
NuGet restore 22s
Starting: NuGet restore
Task : NuGet
Description : Restore, pack, or push NuGet packages, or run a NuGet command. Supports NuGet.org and authenticated feeds like Package Management and MyGet. Uses NuGet.exe and works with .NET Framework apps. For .NET Core and .NET Standard apps, use the .NET Core task.
Version : 2.0.48
Author : Microsoft Corporation
Help : [More Information](https://go.microsoft.com/fwlink/?LinkID=613747)
SYSTEMVSSCONNECTION exists true
SYSTEMVSSCONNECTION exists true
[command]C:\Windows\system32\chcp.com 65001
Active code page: 65001
Detected NuGet version 4.4.1.4656 / 4.4.1
SYSTEMVSSCONNECTION exists true
Saving NuGet.config to a temporary config file.
[command]C:\hostedtoolcache\windows\NuGet\4.4.1\x64\nuget.exe sources Add -NonInteractive -Name NuGetOrg -Source https://api.nuget.org/v3/index.json -ConfigFile D:\a\1\Nuget\tempNuGet_31.config


FunctionsV2-Azure Web App f...
Tasks Variables Triggers Options Retention History
Save & queue Discard Summary Queue ...
Pipeline
Build pipeline
Get sources
FunctionsV2 master
Agent job 1
Run on agent
Use NuGet 4.4.1
NuGet Tool Installer
NuGet restore
NuGet
Build solution ***.sln
Visual Studio Build
VsTest - testAssemblies
Visual Studio Test
Azure App Service Deploy: functionsv2app
Azure App Service Deploy
Publish symbols path
Index Sources & Publish Symbols
Publish Artifact: drop
Publish Build Artifacts
View YAML
Name *
FunctionsV2-Azure Web App for ASP.NET-CI
Agent pool * | Pool information | Manage
Hosted VS2017
Parameters | Unlink all
Solution *
***.sln
Azure subscription * | Manage
Azure Free Trial (0bf166ac-9aa8-4597-bb2a-a845afe01415)
Scoped to resource group 'RG_FunctionsV2'
App service name *
functionsv2app

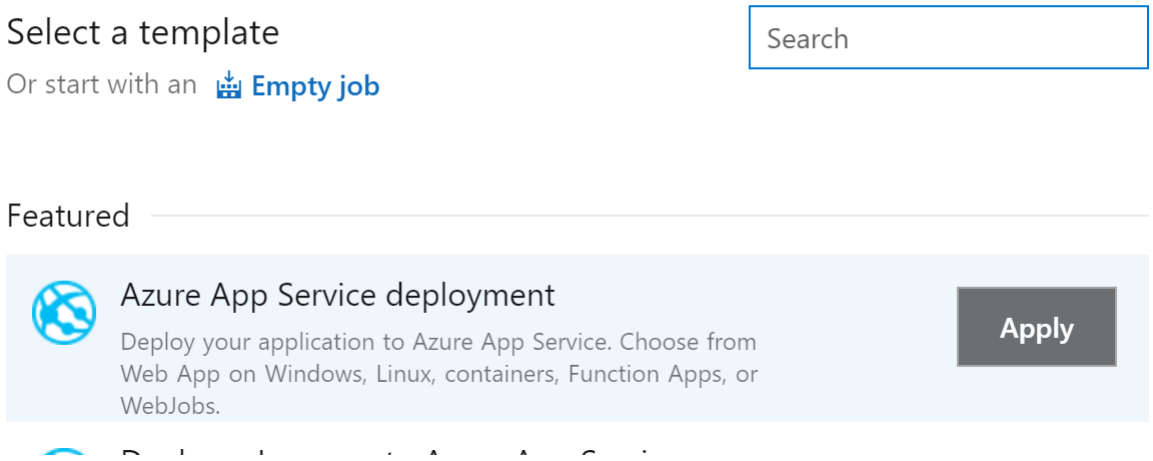
16. Once the build is complete you will get an email.



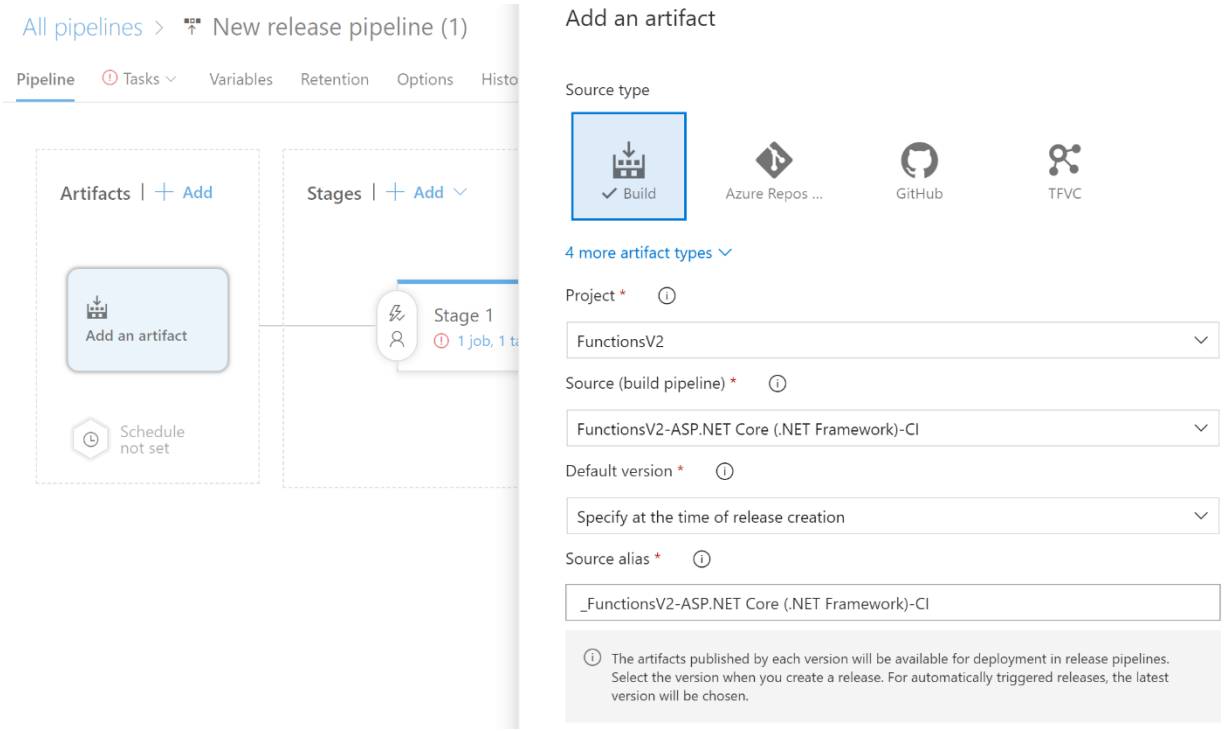
Step 6 - Create and execute a release template

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

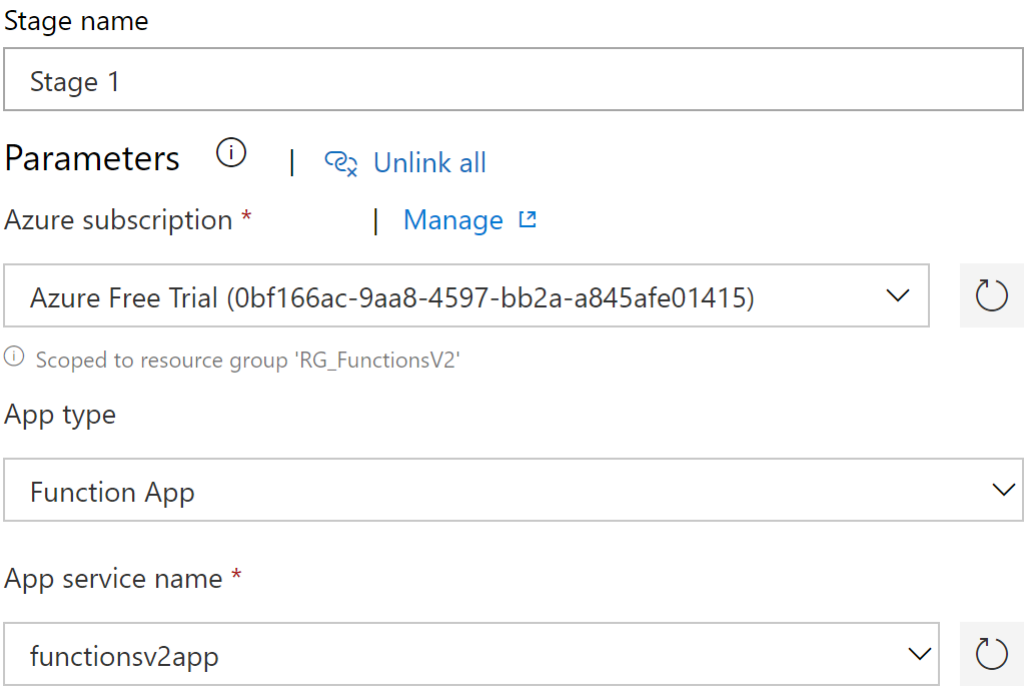
1. In the left hand pane select release pipeline.
2. Click New release pipeline.
3. Choose Azure App Service deployment.



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

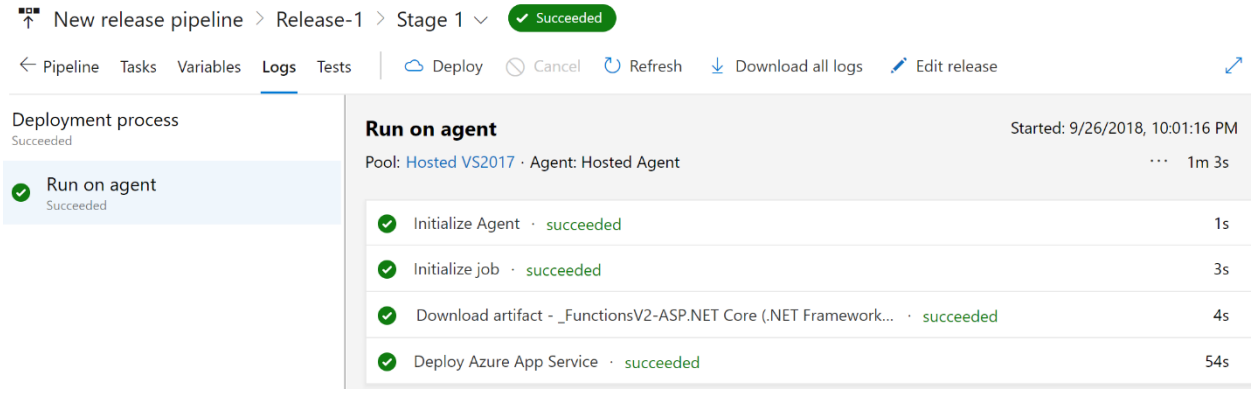


6. Click **Add**.
7. Go to stage 1 and select Deploy Azure App Service Task.
8. Set the parameters to correct Azure Subscription.
9. Set the correct App Type and App Service Name.



10. Create a release.

11. Observe the release progress.




Azure
THURSDays
Every first thursday of the month