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**.

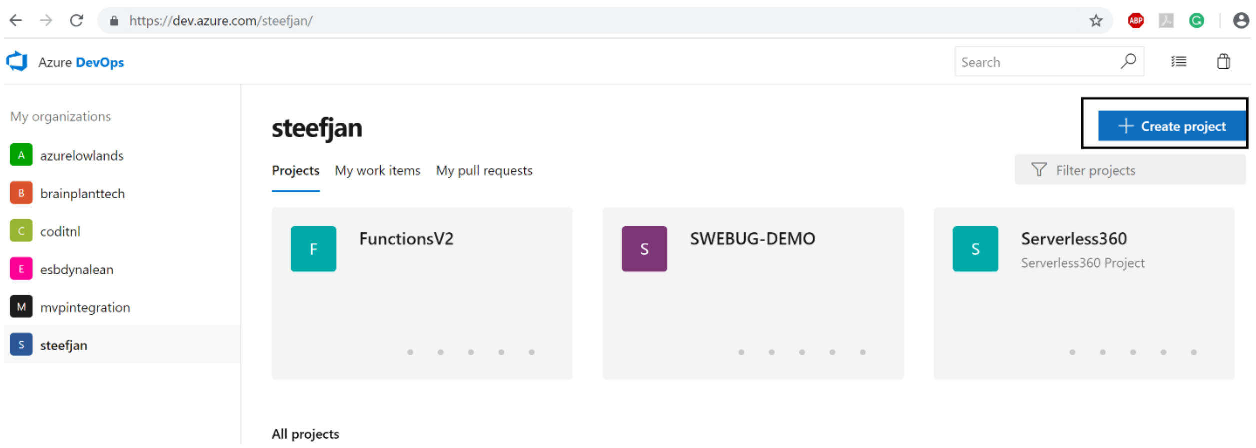
Created by Steef-Jan Wiggers, Microsoft Azure MVP, Codit Domain Lead Azure

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

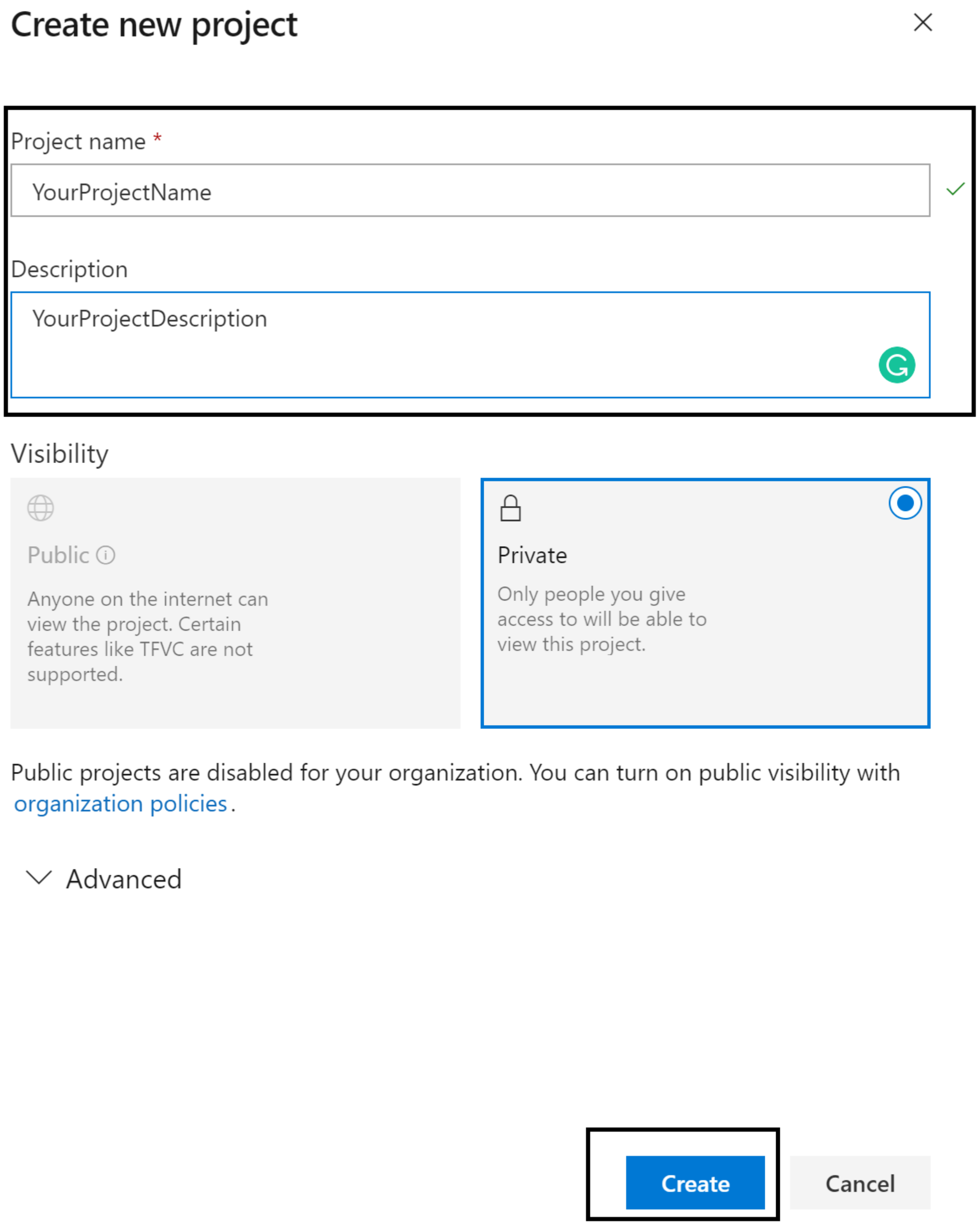
Step 1 - Create an Azure DevOps Project

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**.


Create new project

Project name *

YourProjectName

Description

YourProjectDescription

Visibility

Anyone on the internet can view the project. Certain features like TFVC are not supported.

[✓ Private]

Only people you give access to will be able to view this project.

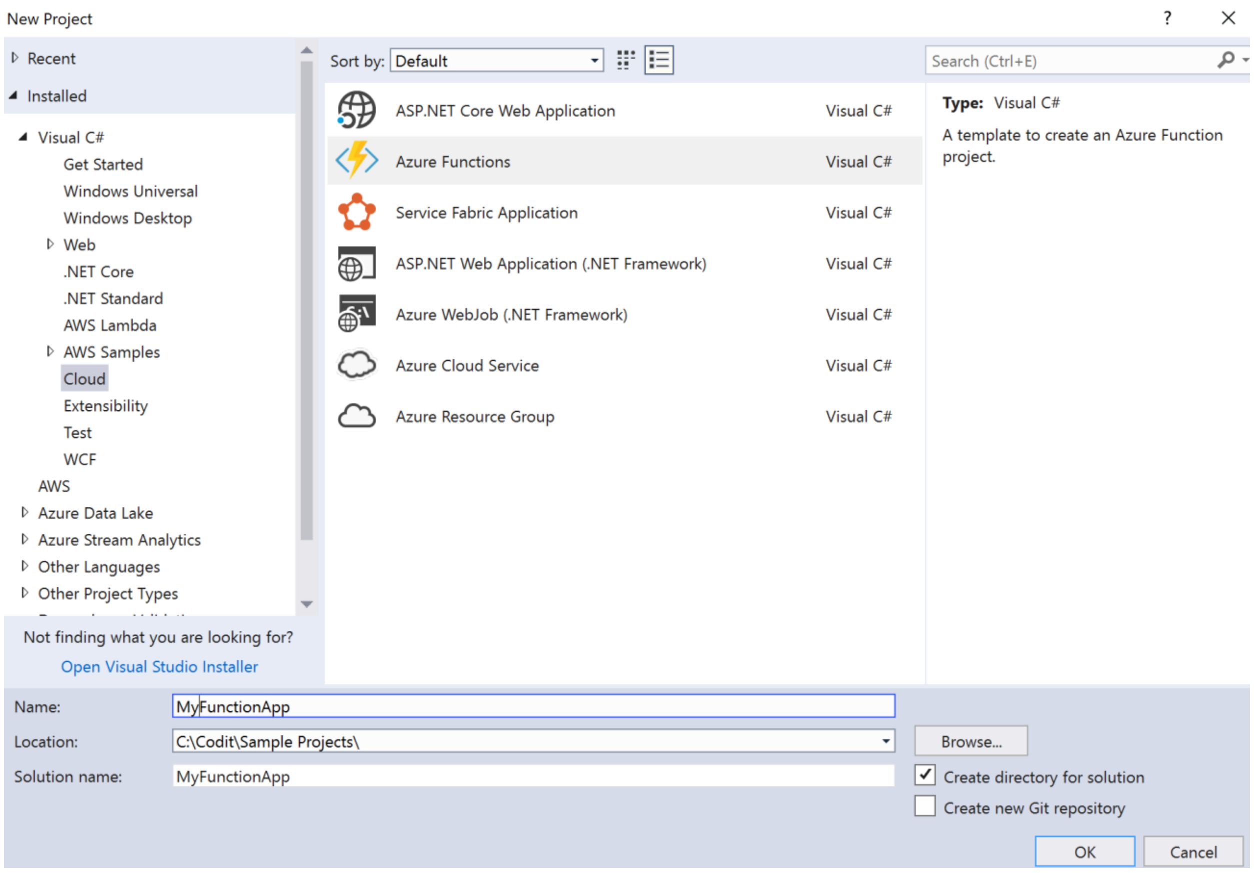
Public projects are disabled for your organization. You can turn on public visibility with organization policies.

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.


New Project

Recent
Installed
Visual C#
Get Started
Windows Universal
Windows Desktop
Web
.NET Core
.NET Standard
AWS Lambda
AWS Samples
Cloud
Extensibility
Test
WCF
AWS
Azure Data Lake
Azure Stream Analytics
Other Languages
Other Project Types

Not finding what you are looking for?
Open Visual Studio Installer

Sort by: Default

ASP.NET Core Web Application	Visual C#
Azure Functions	Visual C#
Service Fabric Application	Visual C#
ASP.NET Web Application (.NET Framework)	Visual C#
Azure WebJob (.NET Framework)	Visual C#
Azure Cloud Service	Visual C#
Azure Resource Group	Visual C#

Search (Ctrl+E)

Type: Visual C#

A template to create an Azure Function project.

Name: MyFunctionApp
Location: C:\Codit\Sample Projects\
Solution name: MyFunctionApp

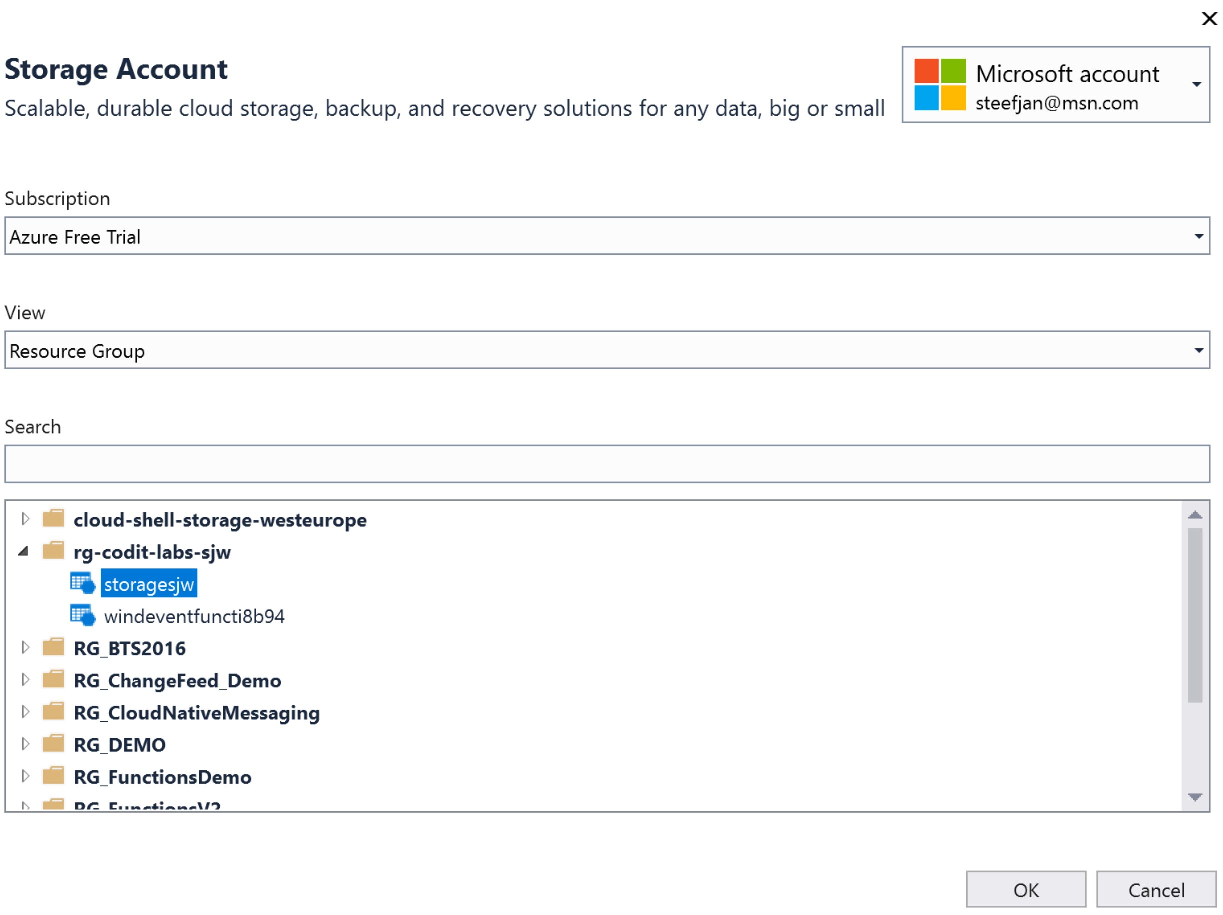
Browse...
[✓] Create directory for solution
[ ] Create new Git repository

OK Cancel

6. Click **Ok**.
7. 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!

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



Storage Account

Scalable, durable cloud storage, backup, and recovery solutions for any data, big or small

Microsoft account
steefjan@msn.com

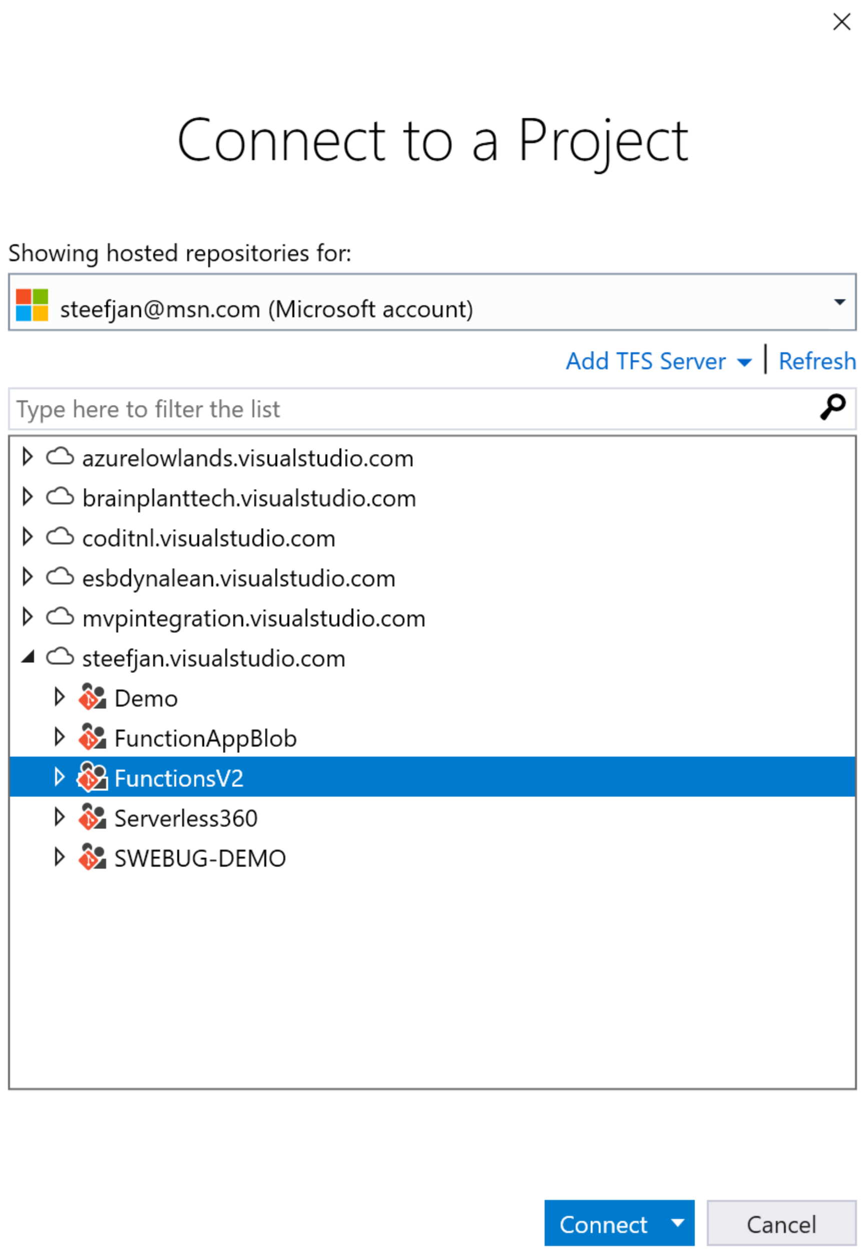
Subscription: Azure Free Trial

View: Resource Group

Search

- cloud-shell-storage-westeurope
- rg-codit-labs-sjw
  - storagesjw
  - windeventfuncti8b94
- RG_BTS2016
- RG_ChangeFeed_Demo
- RG_CloudNativeMessaging
- RG_DEMO
- RG_FunctionsDemo

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**.


Connect to a Project

Showing hosted repositories for:

steefjan@msn.com (Microsoft account)

azurelowlands.visualstudio.com
brainplanttech.visualstudio.com
coditnl.visualstudio.com
esbdynalean.visualstudio.com
mvpintegration.visualstudio.com
steefjan.visualstudio.com
Demo
FunctionAppBlob
FunctionsV2
Serverless360
SWEBUG-DEMO

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; }

    }
```

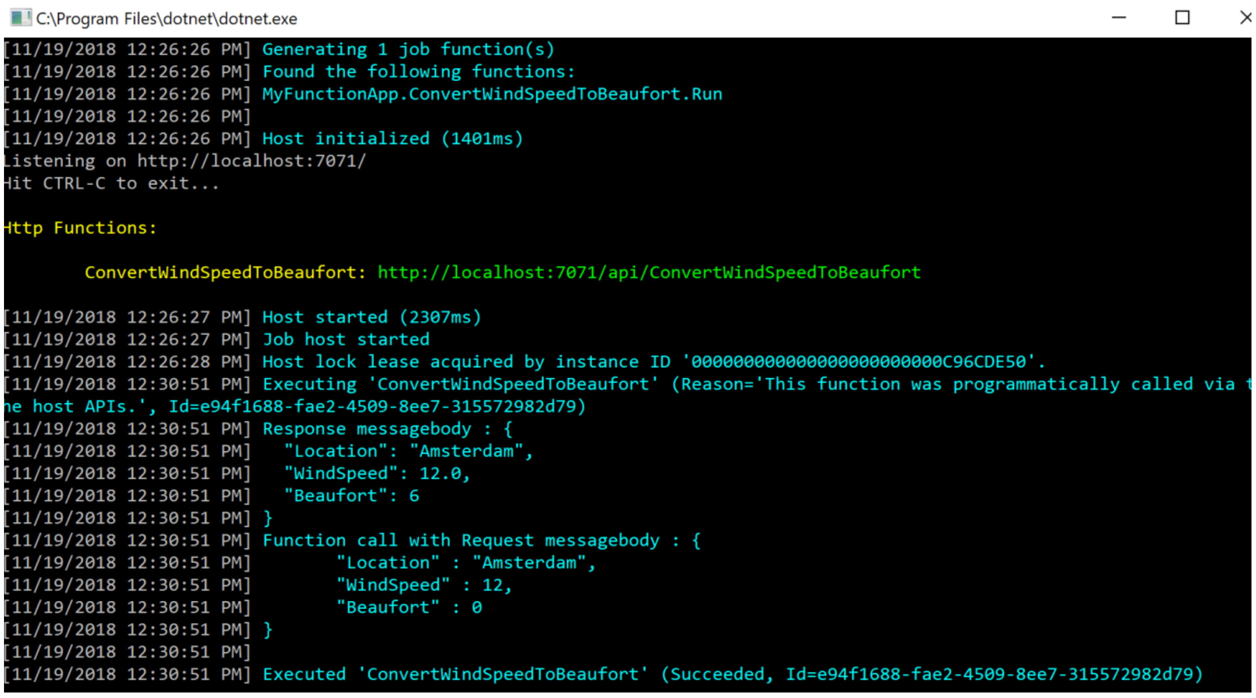
23. Build the project.
24. Run the project.
25. Copy the endpoint http://localhost:7071/api/ConvertWindSpeedToBeaufort
26. Open Postman, and create a POST to the URL
27. Use the following payload:

```
{
        "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.

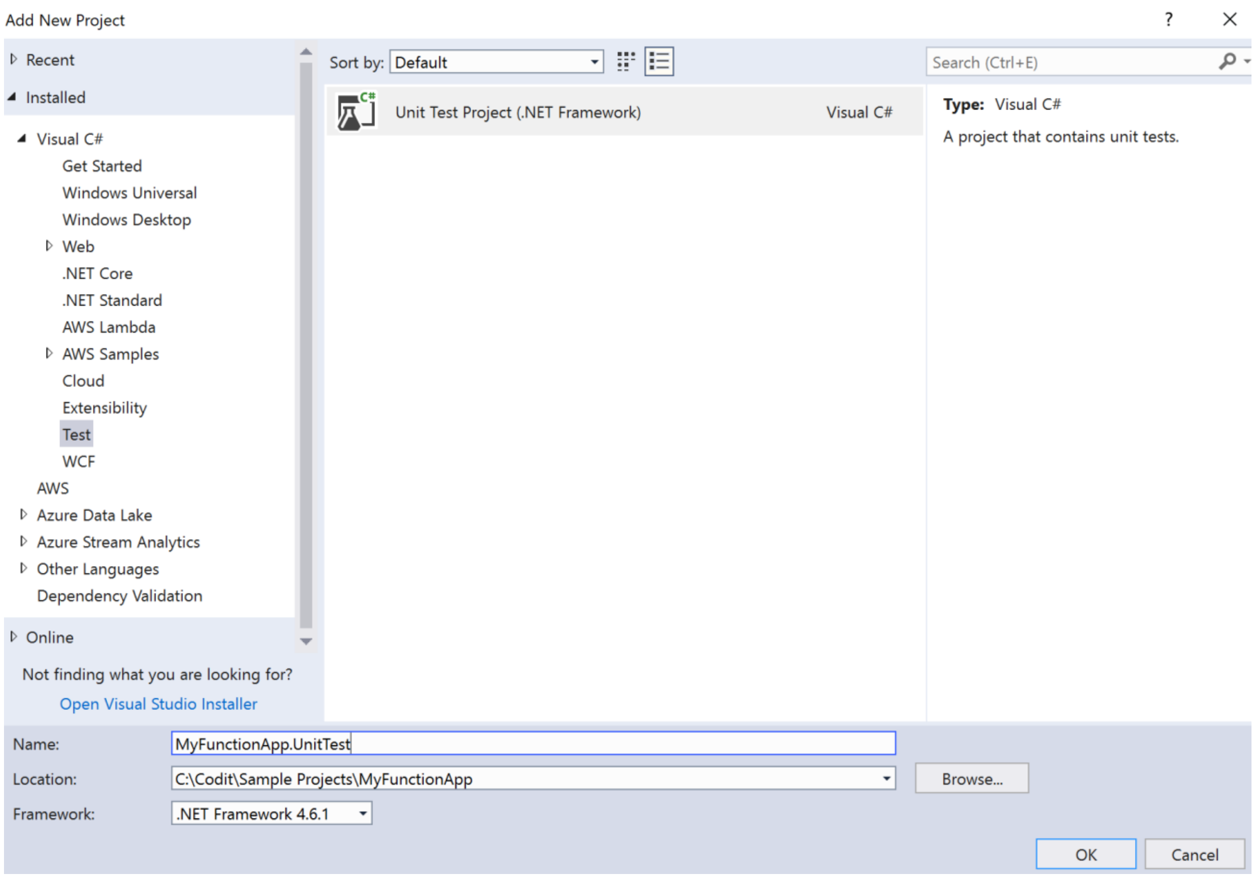

C:\Program Files\dotnet\dotnet.exe

```
[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 '0000000000000000000000000C96CDE50'.
[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.



Type: Visual C#

A project that contains unit tests.

Name: MyFunctionApp.UnitTest

Location: C:\Codit\Sample Projects\MyFunctionApp

Framework: .NET Framework 4.6.1

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]
```

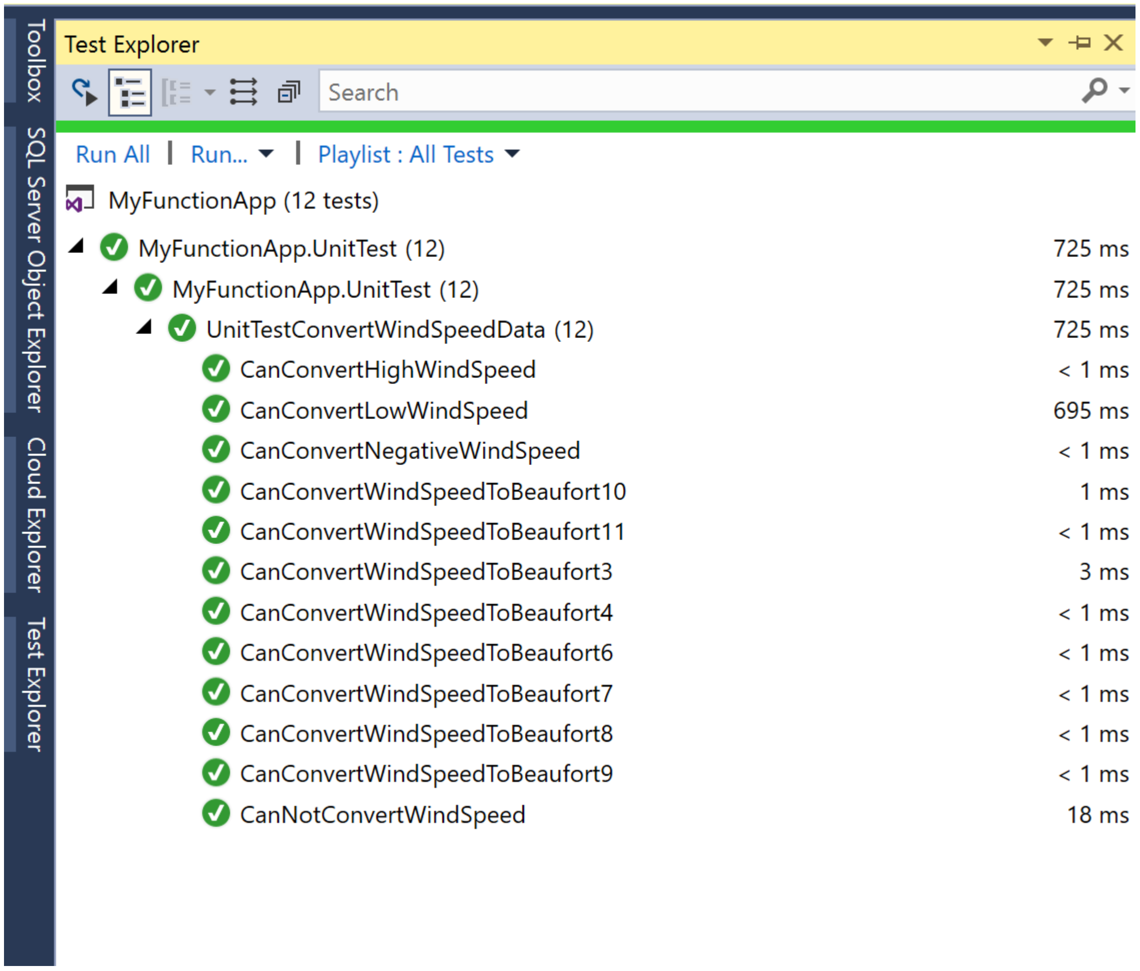
```
public void CanConvertWindSpeedToBeaufort4()
{
  //Arrange
  var windSpeedRequest = new WindSpeedData
  {
    WindSpeed = 7.5,
    Beaufort = 0,
    Location = "Amsterdam"
  };

  //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.

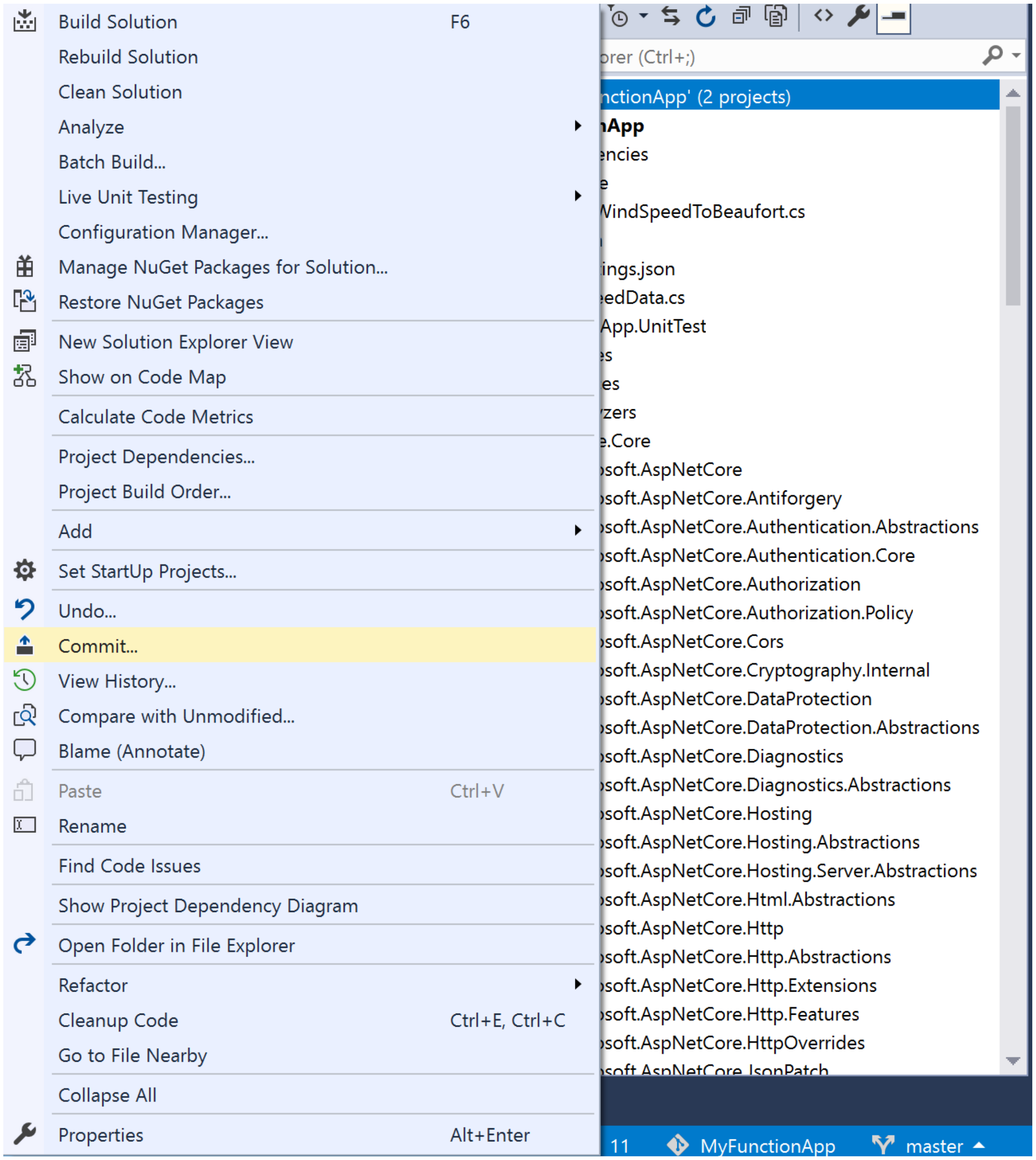

Toolbox
SQL Server Object Explorer
Cloud Explorer
Test Explorer
Test Explorer
Search
Run All | Run... | Playlist : All Tests

```
MyFunctionApp (12 tests)
◢ MyFunctionApp.UnitTest (12)                         725 ms
  ◢ MyFunctionApp.UnitTest (12)                       725 ms
    ◢ UnitTestConvertWindSpeedData (12)               725 ms
        CanConvertHighWindSpeed                       < 1 ms
        CanConvertLowWindSpeed                        695 ms
        CanConvertNegativeWindSpeed                   < 1 ms
        CanConvertWindSpeedToBeaufort10                 1 ms
        CanConvertWindSpeedToBeaufort11               < 1 ms
        CanConvertWindSpeedToBeaufort3                  3 ms
        CanConvertWindSpeedToBeaufort4                < 1 ms
        CanConvertWindSpeedToBeaufort6                < 1 ms
        CanConvertWindSpeedToBeaufort7                < 1 ms
        CanConvertWindSpeedToBeaufort8                < 1 ms
        CanConvertWindSpeedToBeaufort9                < 1 ms
        CanNotConvertWindSpeed                         18 ms
```

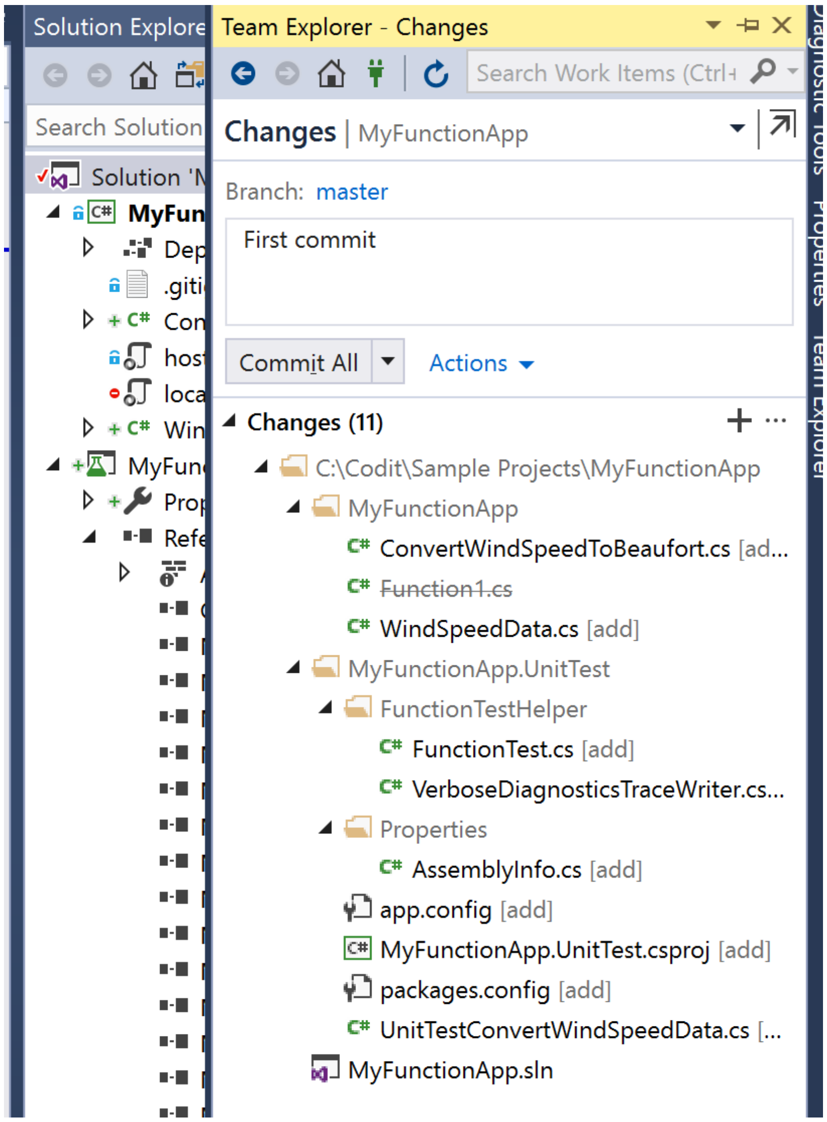
Step 4 – Push the project to Azure DevOps

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.


Build Solution F6
Rebuild Solution
Clean Solution
Analyze
Batch Build...
Live Unit Testing
Configuration Manager...
Manage NuGet Packages for Solution...
Restore NuGet Packages
New Solution Explorer View
Show on Code Map
Calculate Code Metrics
Project Dependencies...
Project Build Order...
Add
Set StartUp Projects...
Undo...
Commit...
View History...
Compare with Unmodified...
Blame (Annotate)
Paste Ctrl+V
Rename
Find Code Issues
Show Project Dependency Diagram
Open Folder in File Explorer
Refactor
Cleanup Code Ctrl+E, Ctrl+C
Go to File Nearby
Collapse All
Properties Alt+Enter
orer (Ctrl+;)
nctionApp' (2 projects)
nApp
encies
e
WindSpeedToBeaufort.cs
ings.json
edData.cs
App.UnitTest
es
es
zers
e.Core
osoft.AspNetCore
osoft.AspNetCore.Antiforgery
osoft.AspNetCore.Authentication.Abstractions
osoft.AspNetCore.Authentication.Core
osoft.AspNetCore.Authorization
osoft.AspNetCore.Authorization.Policy
osoft.AspNetCore.Cors
osoft.AspNetCore.Cryptography.Internal
osoft.AspNetCore.DataProtection
osoft.AspNetCore.DataProtection.Abstractions
osoft.AspNetCore.Diagnostics
osoft.AspNetCore.Diagnostics.Abstractions
osoft.AspNetCore.Hosting
osoft.AspNetCore.Hosting.Abstractions
osoft.AspNetCore.Hosting.Server.Abstractions
osoft.AspNetCore.Html.Abstractions
osoft.AspNetCore.Http
osoft.AspNetCore.Http.Abstractions
osoft.AspNetCore.Http.Extensions
osoft.AspNetCore.Http.Features
osoft.AspNetCore.HttpOverrides
11
MyFunctionApp
master

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



First commit

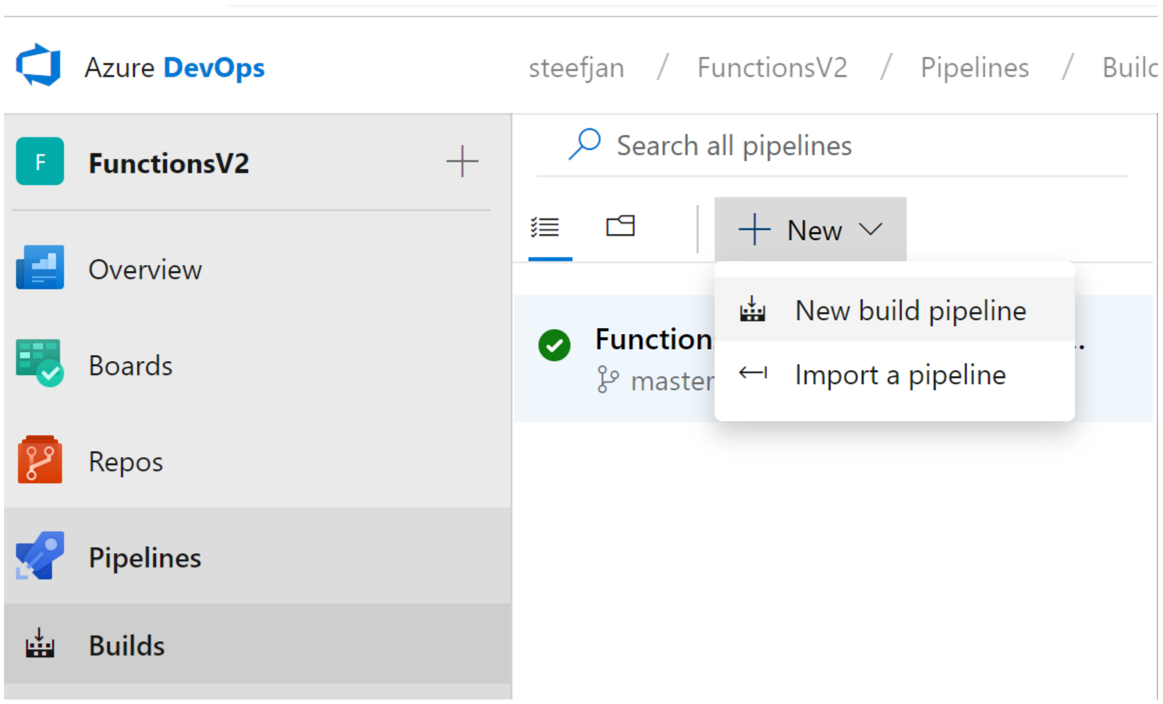
```
C:\Codit\Sample Projects\MyFunctionApp
  MyFunctionApp
    ConvertWindSpeedToBeaufort.cs [ad...
    ~~Function1.cs~~
    WindSpeedData.cs [add]
  MyFunctionApp.UnitTest
    FunctionTestHelper
      FunctionTest.cs [add]
      VerboseDiagnosticsTraceWriter.cs...
    Properties
      AssemblyInfo.cs [add]
    app.config [add]
    MyFunctionApp.UnitTest.csproj [add]
    packages.config [add]
    UnitTestConvertWindSpeedData.cs [...
  MyFunctionApp.sln
```

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.

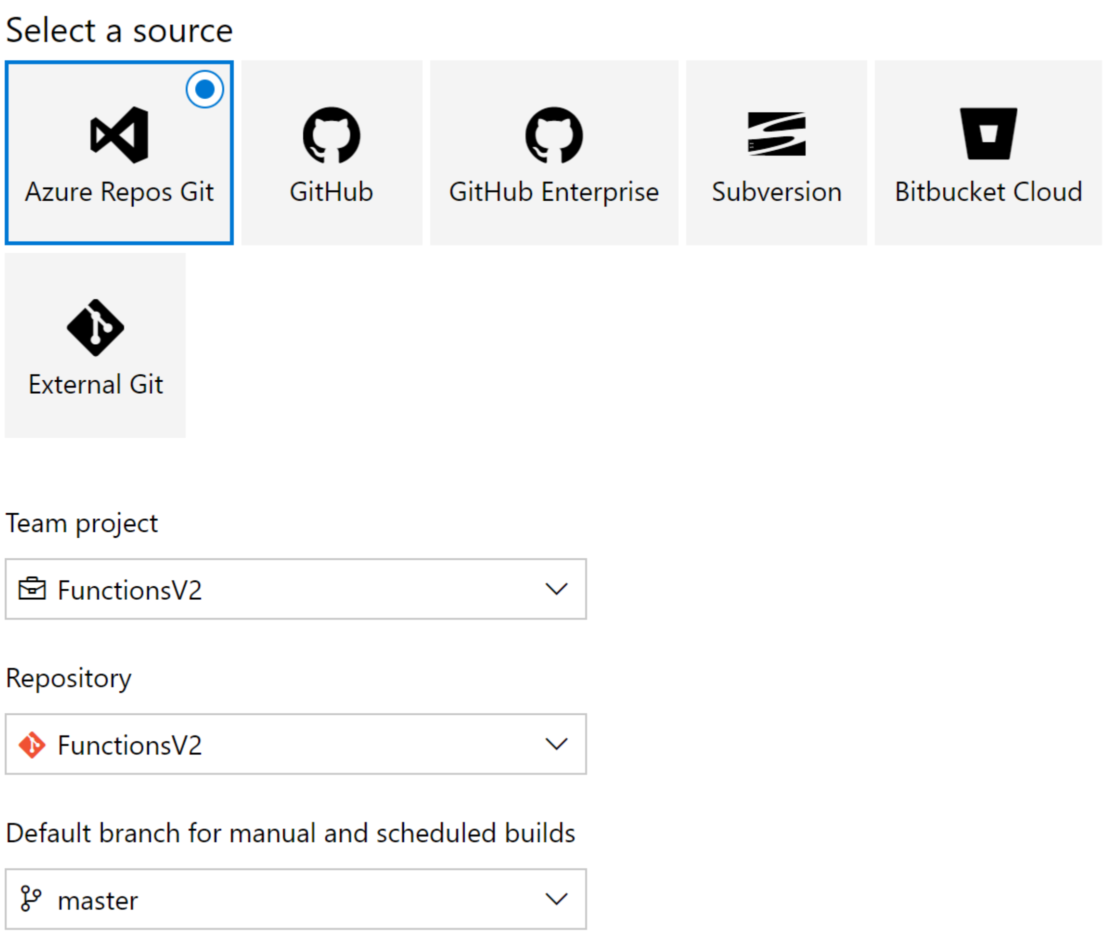
Step 5 - Create and execute a build template

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



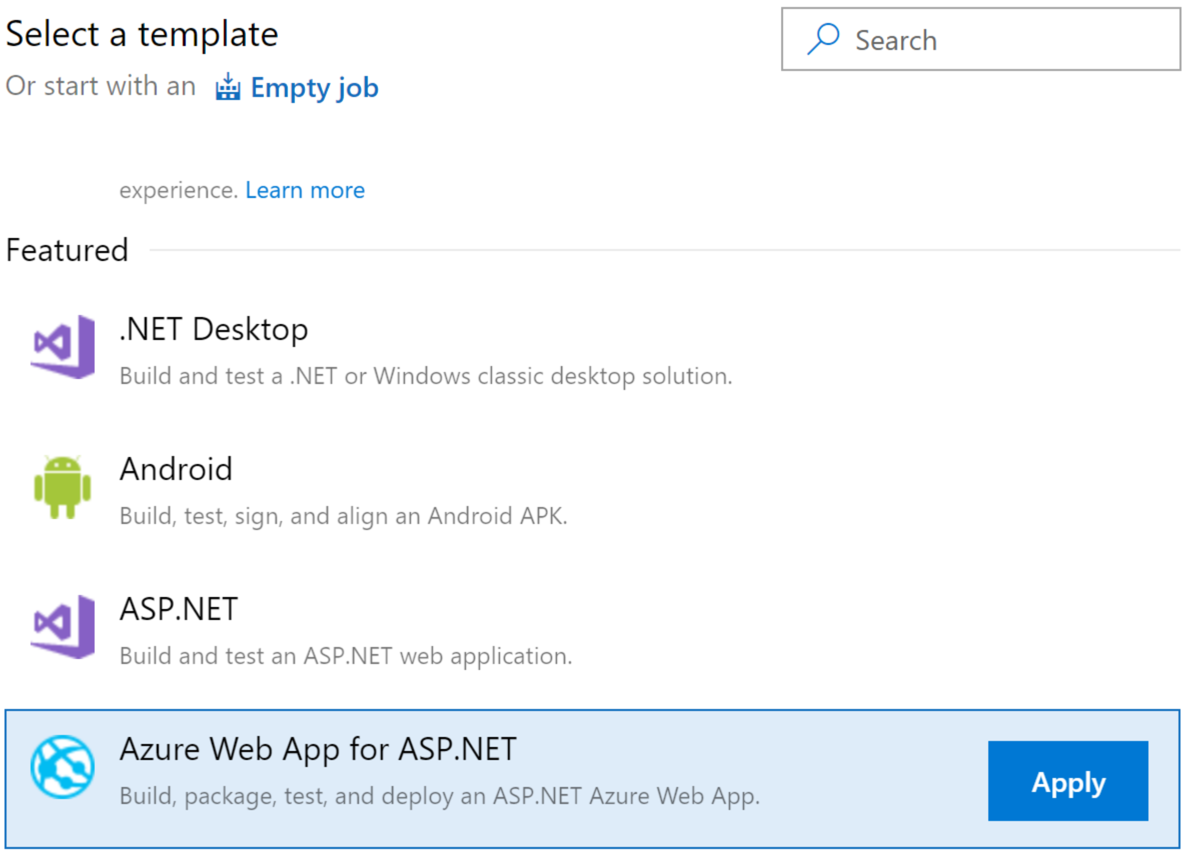
Select a source

Team project: FunctionsV2

Repository: FunctionsV2

Default branch for manual and scheduled builds: master

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


Select a template

Or start with an Empty job

experience. Learn more

Featured

.NET Desktop
Build and test a .NET or Windows classic desktop solution.

Android
Build, test, sign, and align an Android APK.

ASP.NET
Build and test an ASP.NET web application.

Azure Web App for ASP.NET
Build, package, test, and deploy an ASP.NET Azure Web App.

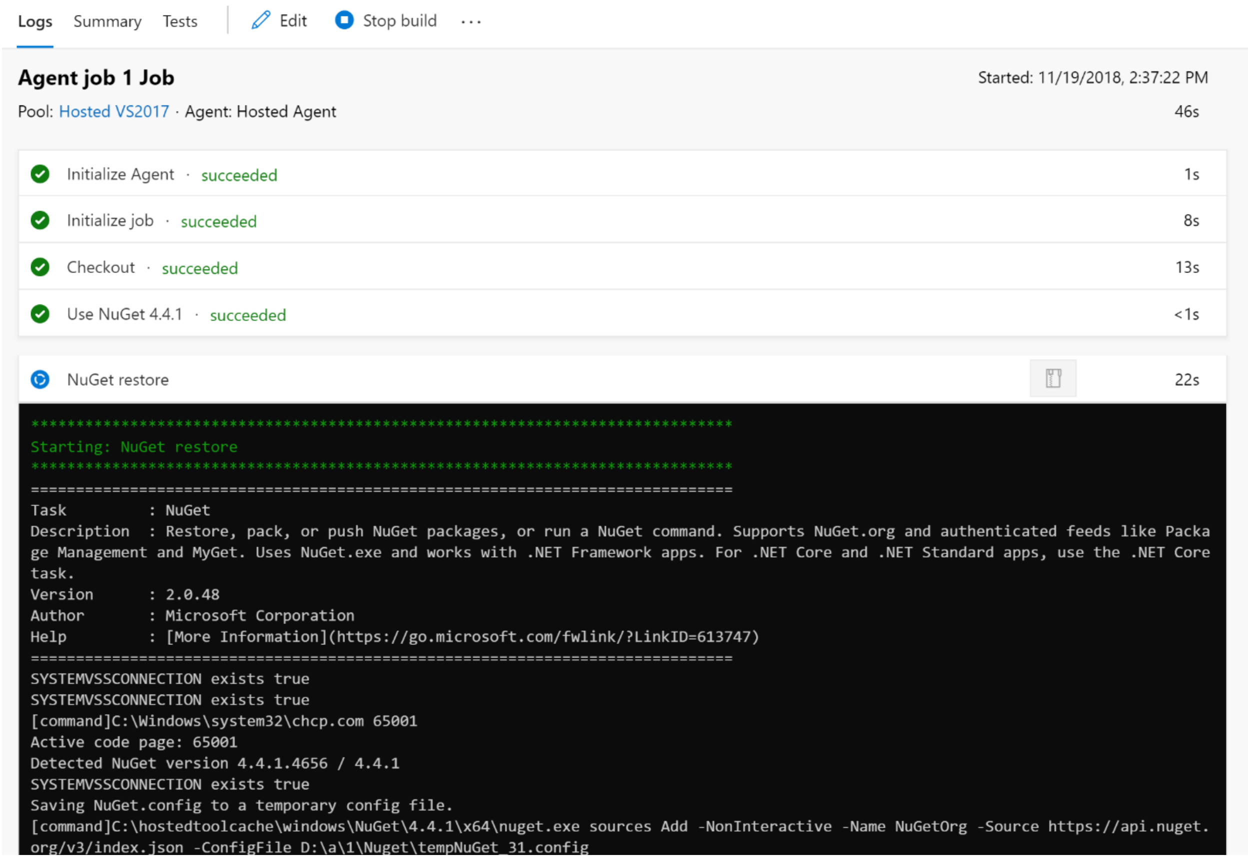
7. In Azure Subscription click Manage
8. Now you will have to create a managed connection i.e. service principle. You can do this from DevOps portal.
9. This connection is necessary to have appropiate right for the build template.
10. In the template set the connection to the correct Azure Subscription 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.



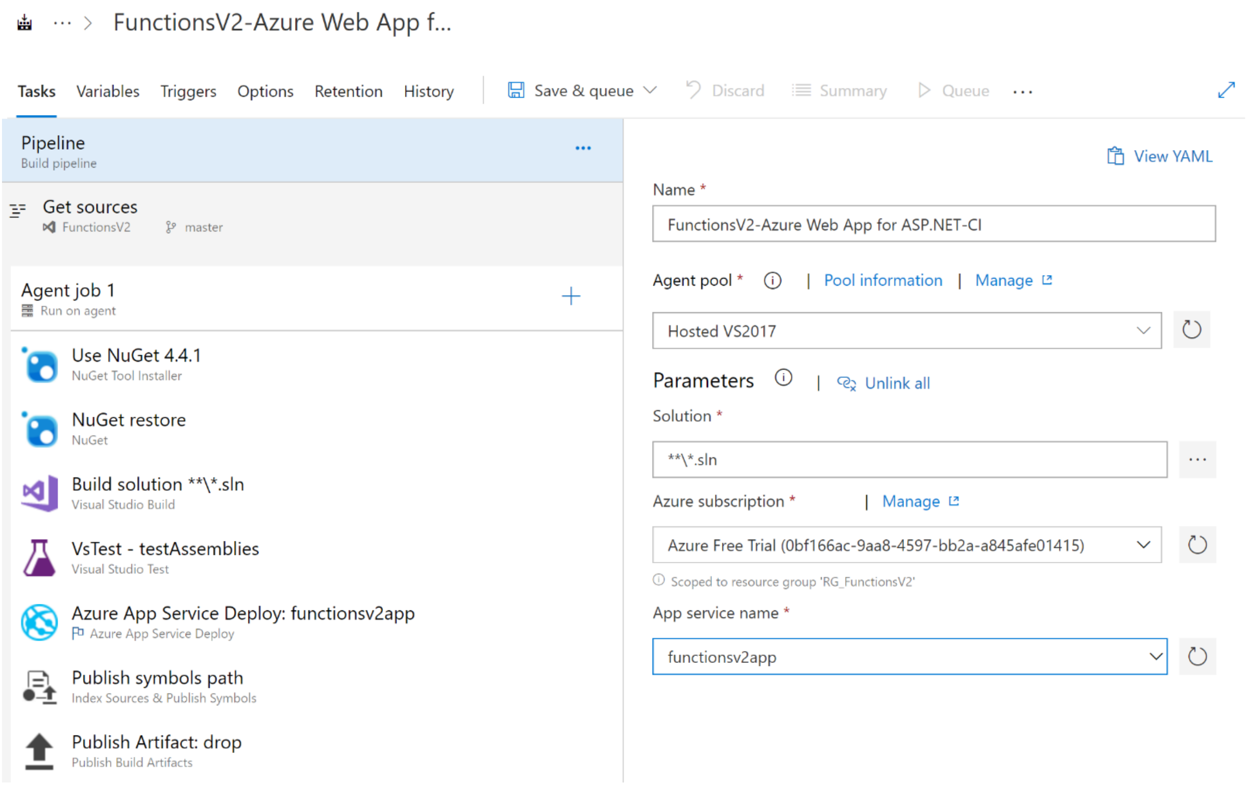
Agent job 1 Job

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 Packa
ge 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
```



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

Name *: FunctionsV2-Azure Web App for ASP.NET-CI

Agent pool * | Pool information | Manage: Hosted VS2017

Parameters | Unlink all

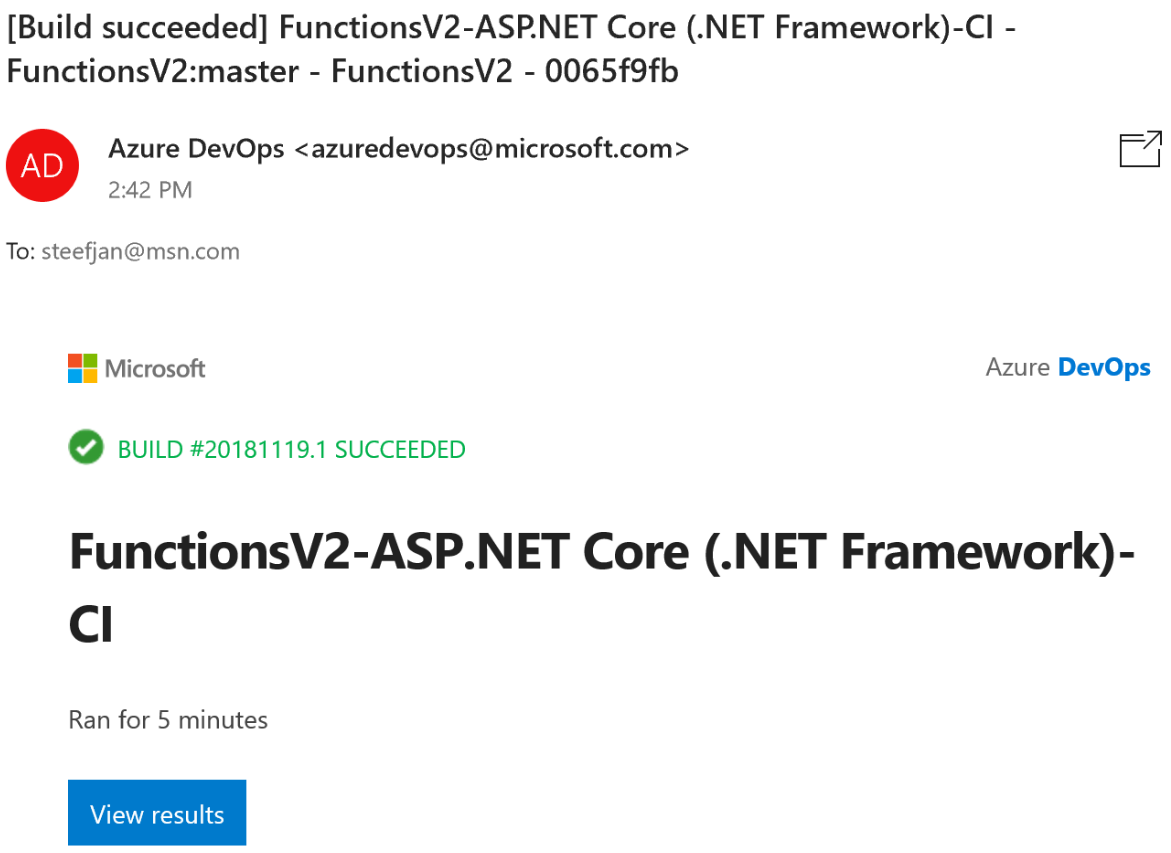
Solution *: ***.sln

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.



[Build succeeded] FunctionsV2-ASP.NET Core (.NET Framework)-CI - FunctionsV2:master - FunctionsV2 - 0065f9fb

Azure DevOps <azuredevops@microsoft.com>
2:42 PM

To: steefjan@msn.com

BUILD #20181119.1 SUCCEEDED

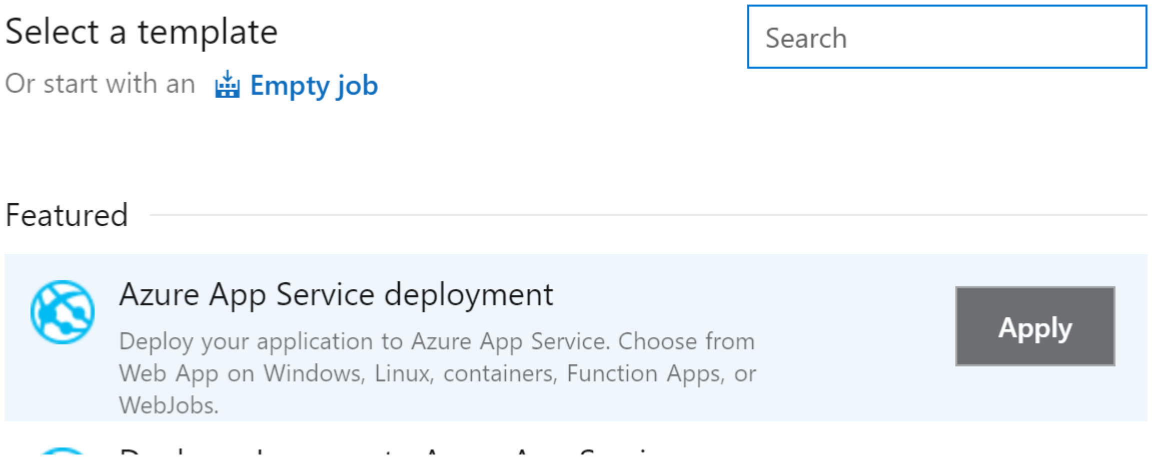
FunctionsV2-ASP.NET Core (.NET Framework)-CI

Ran for 5 minutes

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.


Select a template

Or start with an Empty job

Search

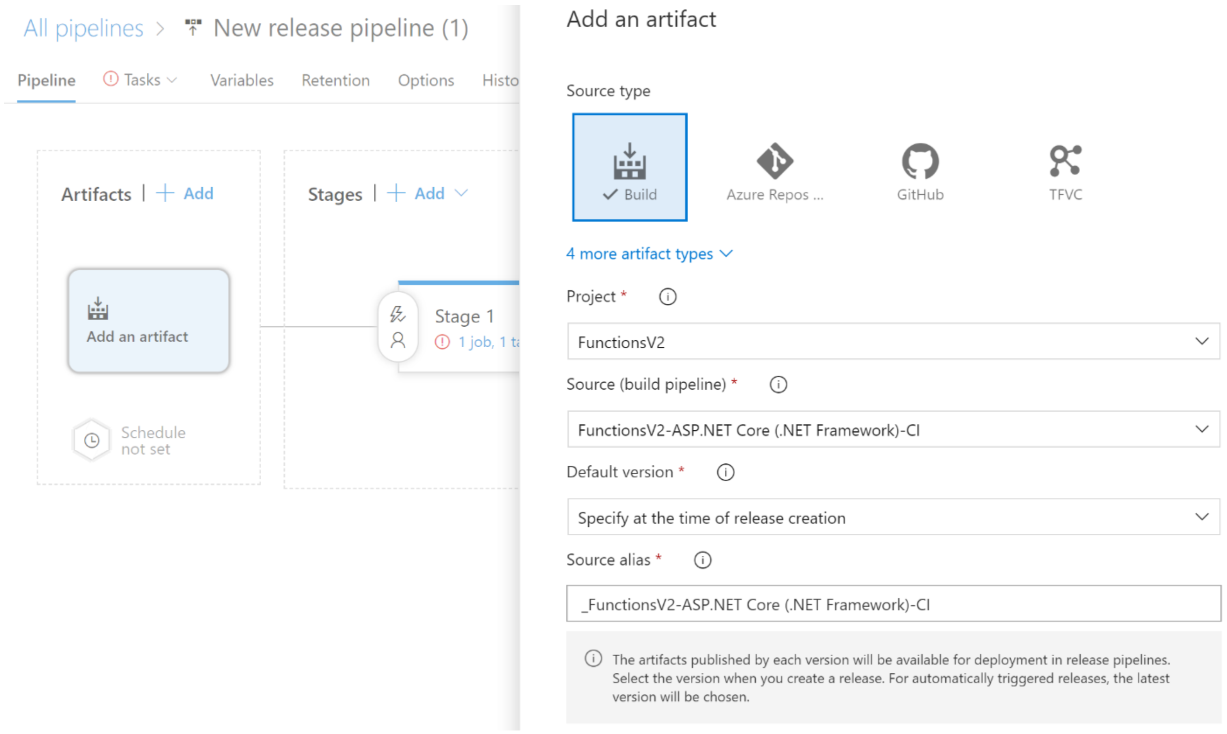
Featured

Azure App Service deployment

Deploy your application to Azure App Service. Choose from Web App on Windows, Linux, containers, Function Apps, or WebJobs.

Apply

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


All pipelines > New release pipeline (1)

Pipeline | Tasks | Variables | Retention | Options | Histo

Artifacts | + Add

Add an artifact

Schedule not set

Stages | + Add

Stage 1

1 job, 1 ta

Add an artifact

Source type

Build | Azure Repos ... | GitHub | TFVC

4 more artifact types

Project *

FunctionsV2

Source (build pipeline) *

FunctionsV2-ASP.NET Core (.NET Framework)-CI

Default version *

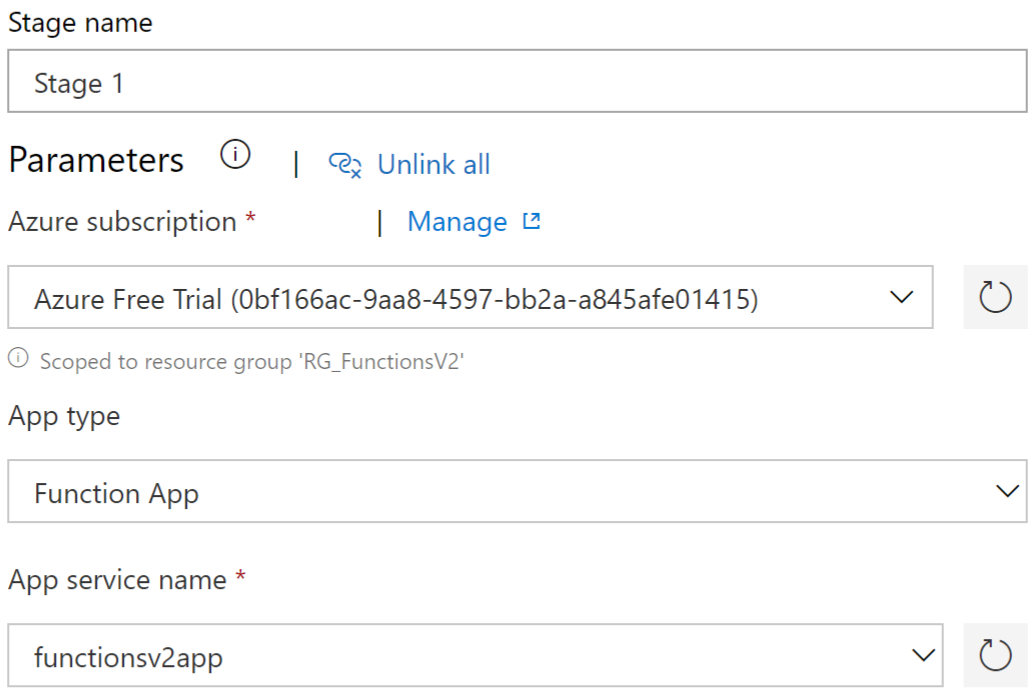
Specify at the time of release creation

Source alias *

_FunctionsV2-ASP.NET Core (.NET Framework)-CI

The artifacts published by each version will be available for deployment in release pipelines. Select the version when you create a release. For automatically triggered releases, the latest version will be chosen.

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.


Stage name

Stage 1

Parameters | Unlink all

Azure subscription * | Manage

Azure Free Trial (0bf166ac-9aa8-4597-bb2a-a845afe01415)

Scoped to resource group 'RG_FunctionsV2'

App type

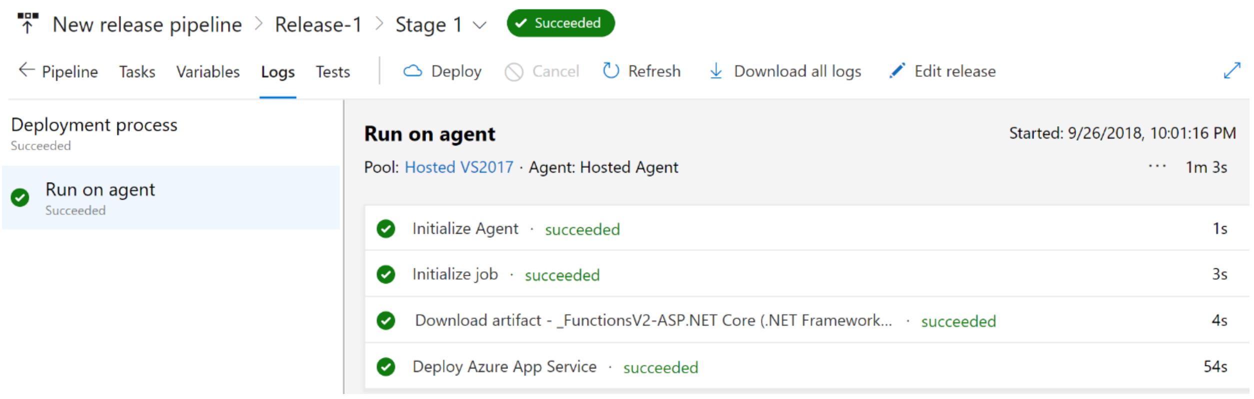
Function App

App service name *

functionsv2app

10. Create a release.

11. Observe the release progress.



New release pipeline > Release-1 > Stage 1 ✓ Succeeded

Deployment process
Succeeded

✓ Run on agent
Succeeded

Run on agent
Started: 9/26/2018, 10:01:16 PM
Pool: Hosted VS2017 · Agent: Hosted Agent ··· 1m 3s

✓	Initialize Agent · succeeded	1s
✓	Initialize job · succeeded	3s
✓	Download artifact - _FunctionsV2-ASP.NET Core (.NET Framework... · succeeded	4s
✓	Deploy Azure App Service · succeeded	54s