**Getting Started With**

**Microsoft Visual Studio Team Services (VSTS)**

**Part 2**

By

V Niranjan

# Introduction

In **Part 1** of the series on Microsoft VSTS we saw how to enable a account, create a project in Microsoft Visual Stuio.Net and share it with VSTS repo.

In this **part 2** article we will extend the same to show case **Devops** features especially **Continuous Integration and Continuous Delivery** to an **Azure Web App**. Apart from deploying as an Azure Web App you can also deploy to a Windows Virtual Machine (VM) which is created as an instance from Azure portal.

In this article we will see the following:

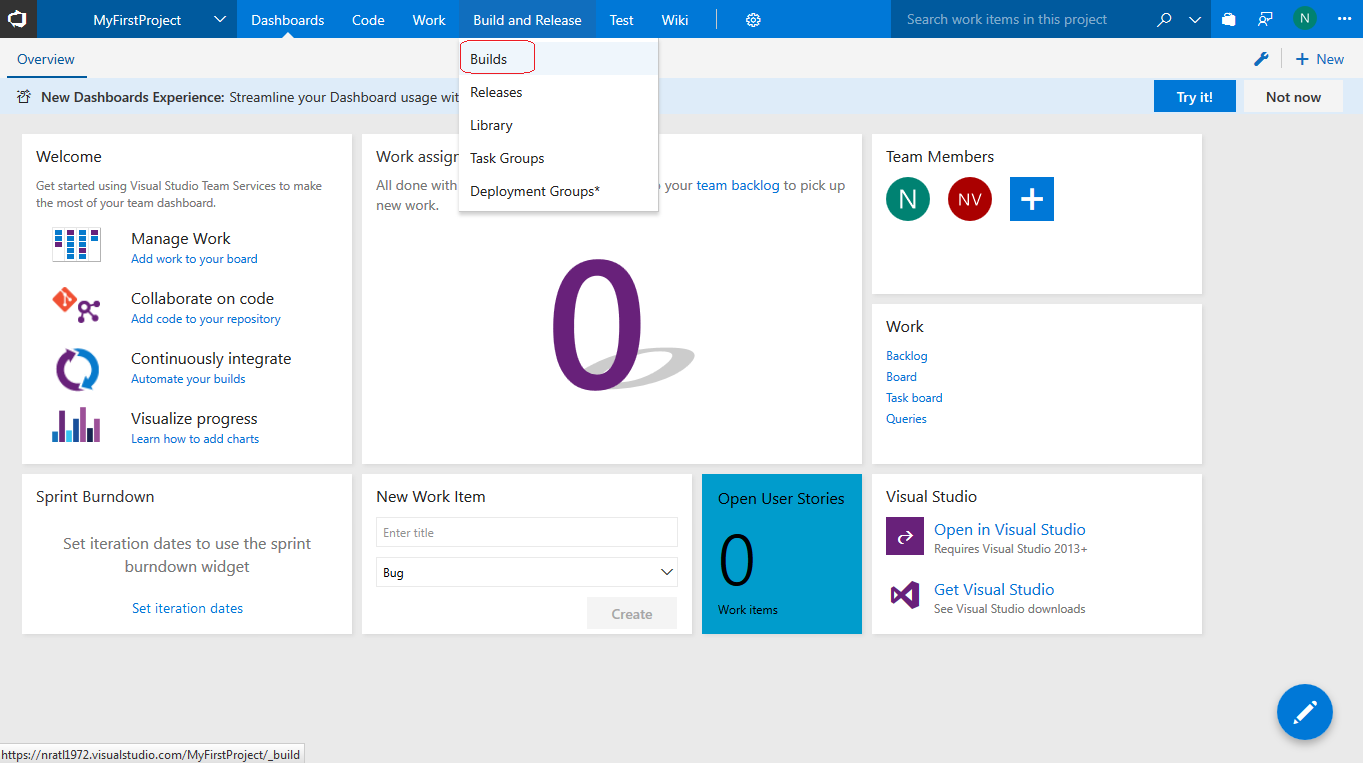
* Create Build Definition ( Continuous Integration )
* Create Release Definition ( Continuous Delivery )

# Create Build Definition (Continuous Integration)

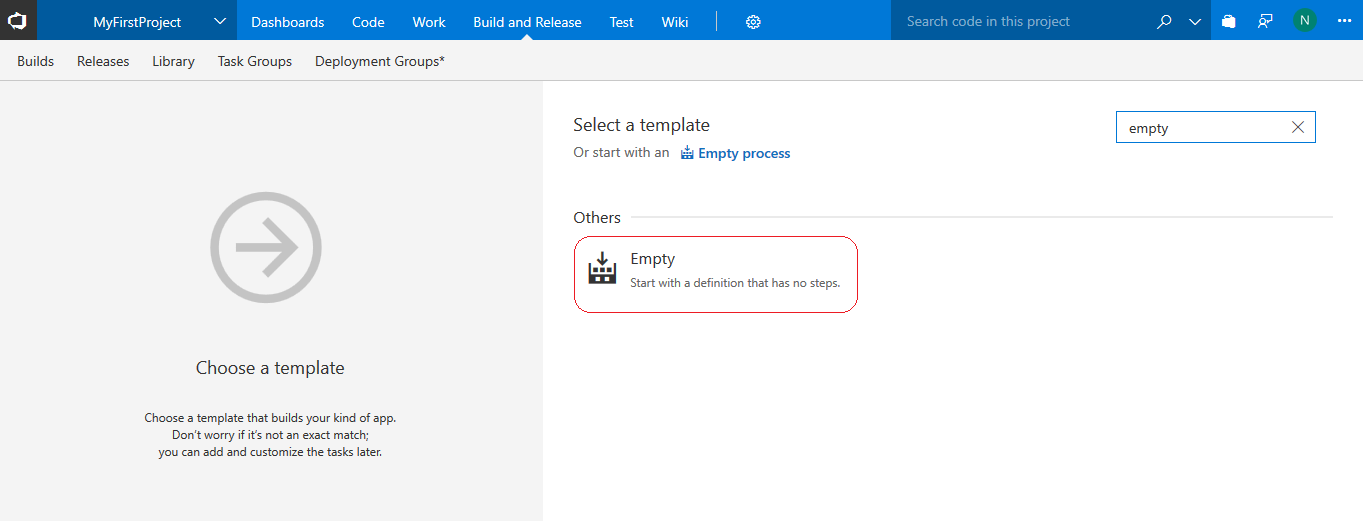
A continuous integration or CI is a process when every time a developer commits or delivers source code changes to the version control repository in this case it is the VSTS repo the build is performed either on every check-in or using a schedule. To start setting up continuous integration in VSTS we first start by creating a build definition.

**Create a new build definition**

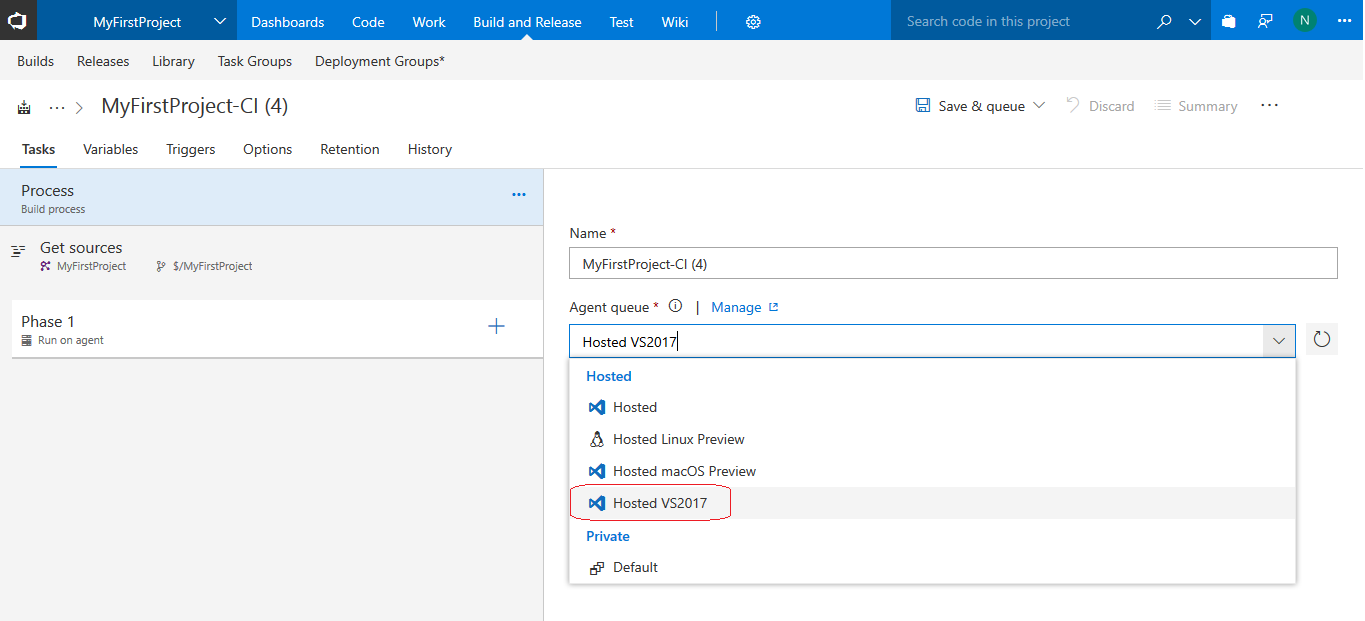
Go to the **Builds section of Build and Release hub** and create a **New definition**

****

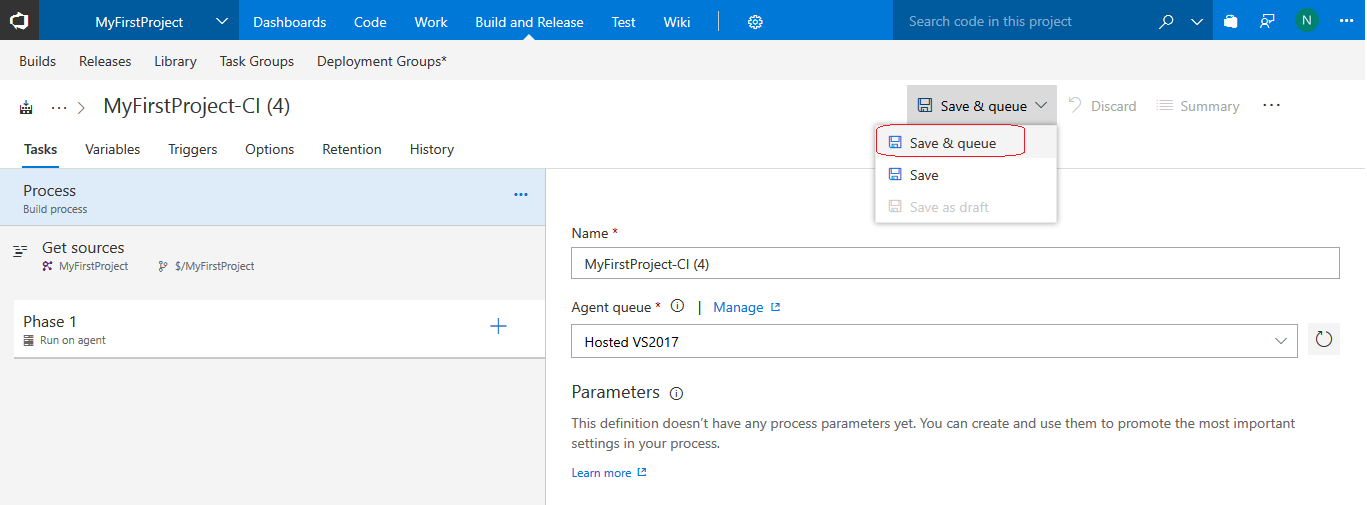
Click on **Empty Process** template to start. We will add all the steps that is needed to build our solution successfully.

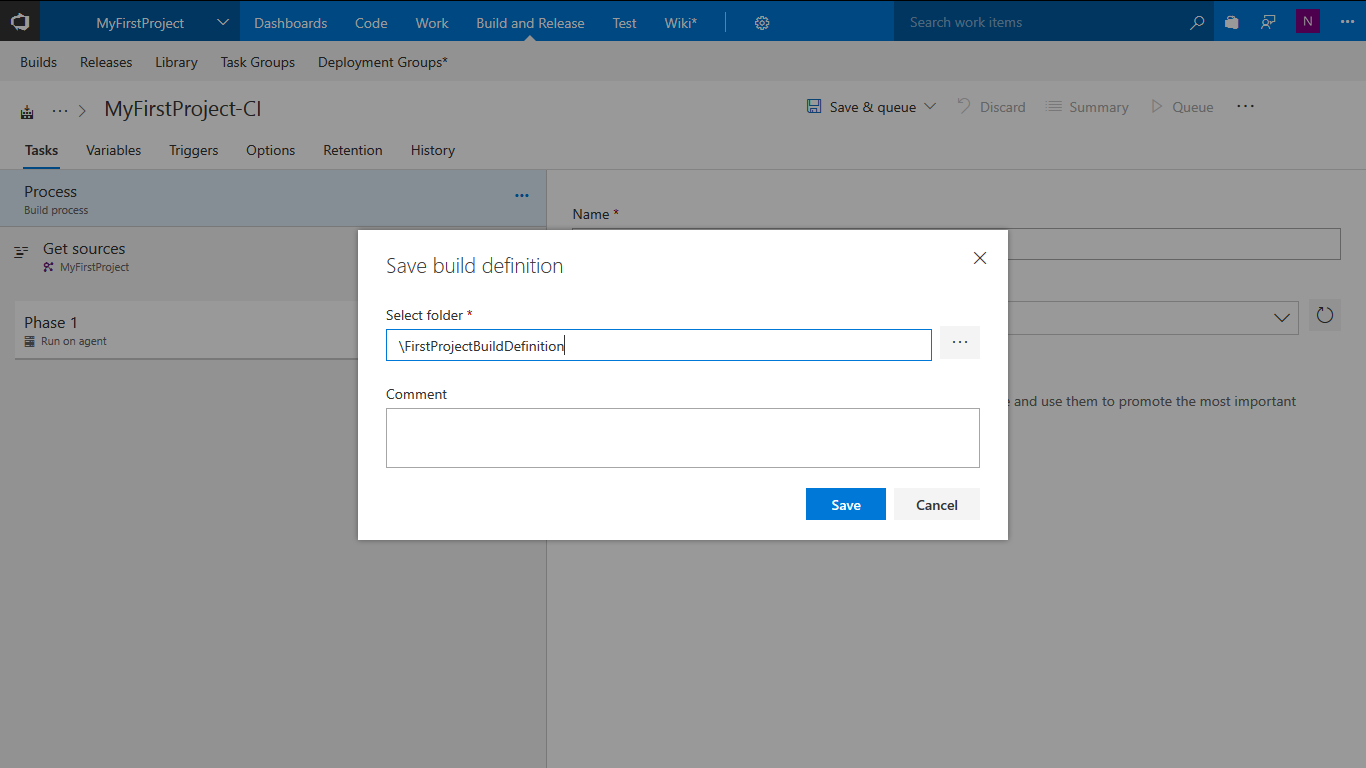


Agents are normally required on the build machines and the environments where the build and deploy would be done. So here in VSTS this agent queue is inbuilt for you to build your app. This is very comfortable to use as you do not need to download and install any agents for build and release. In the first step select Agent queue as Hosted 2017.

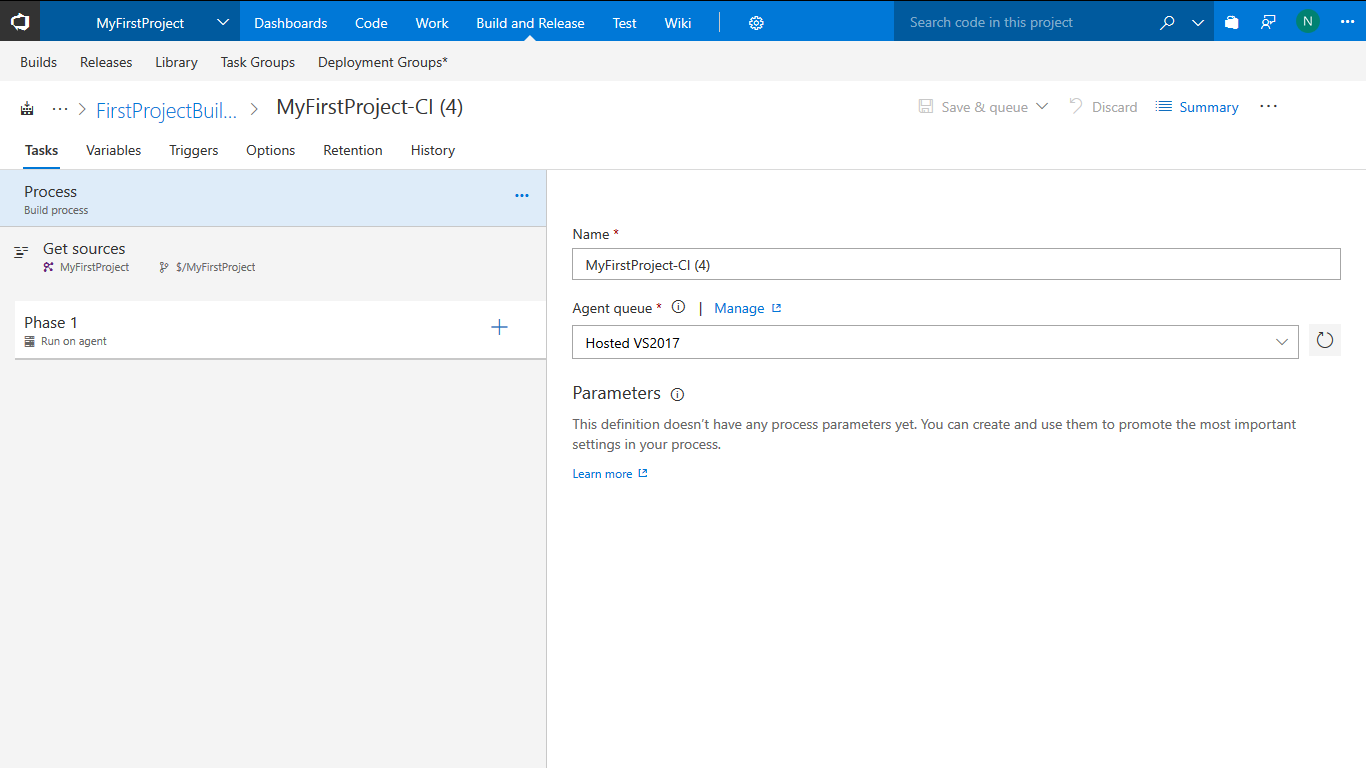


Make sure to save the Build definition.



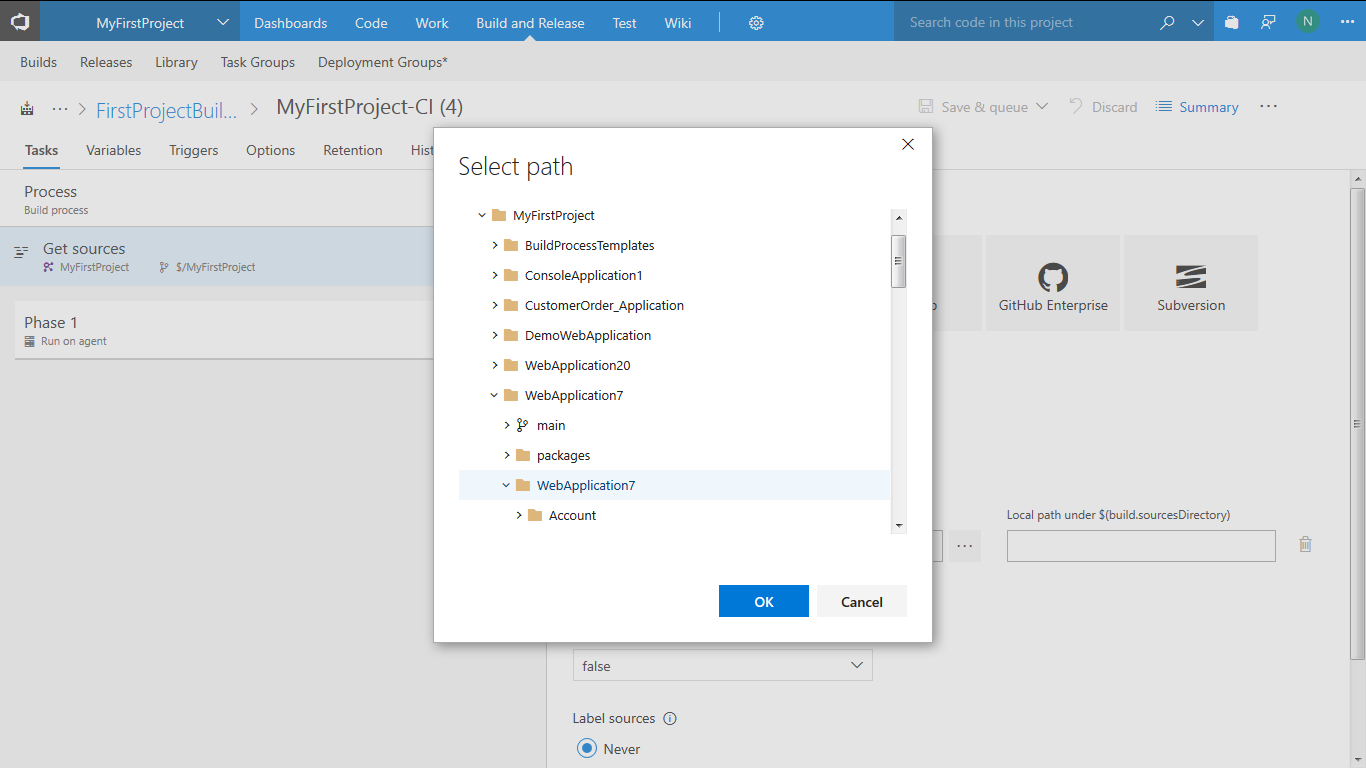


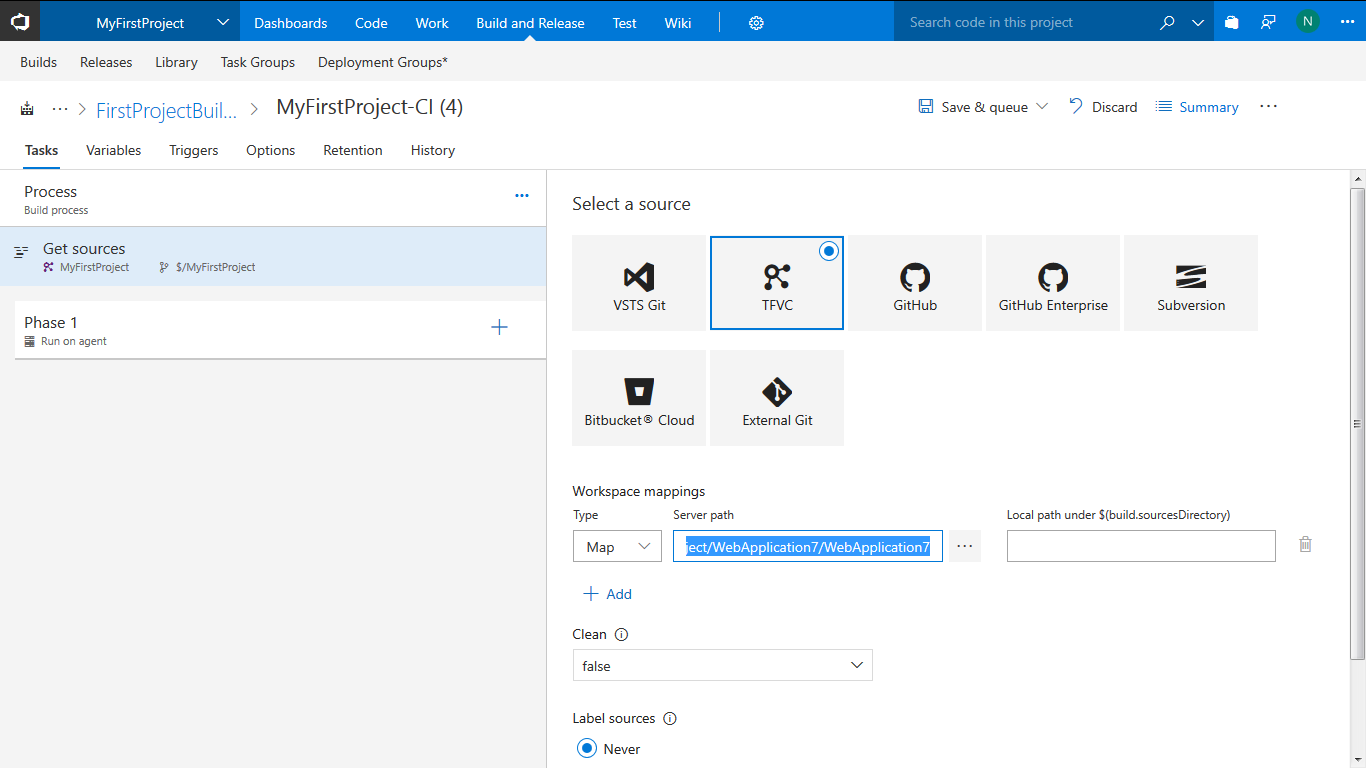
Click on **Save**



Click on **Get Sources** task to select the visual studio solution which is shared to VSTS version control repository

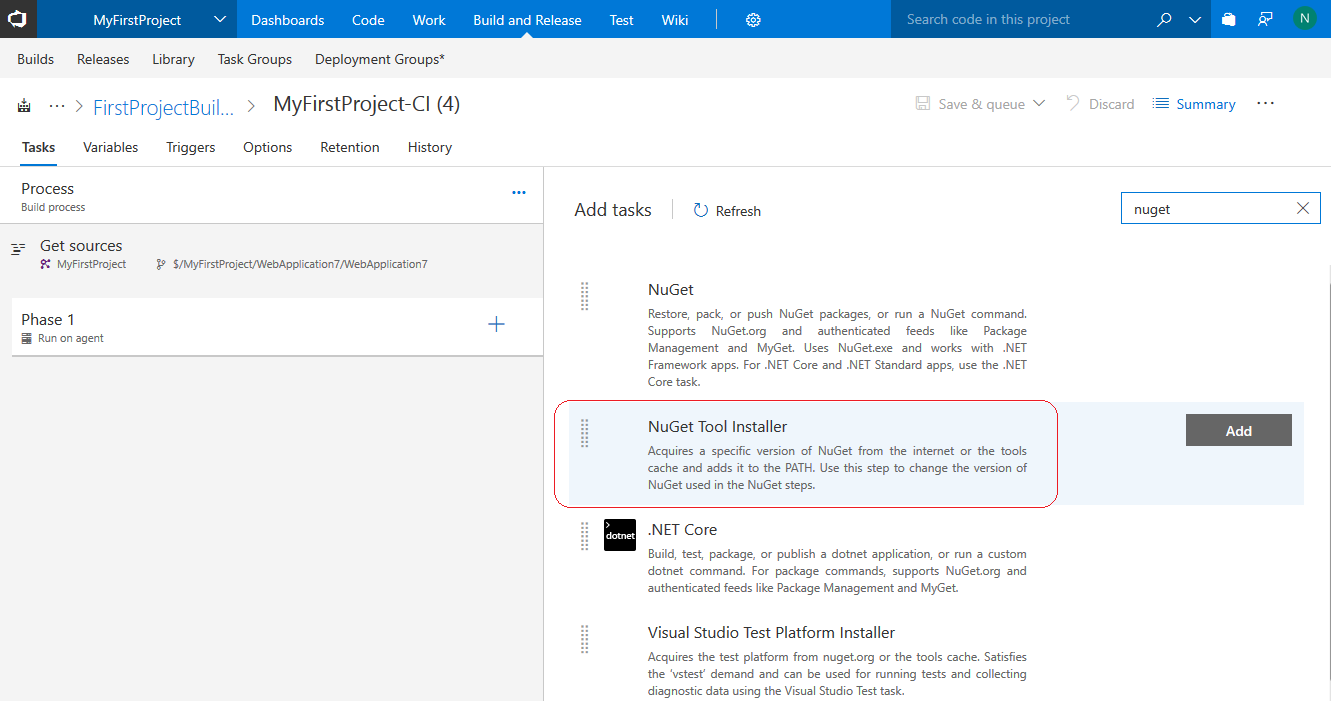
Click on 3 DOTS (…) next to Server path and select the solution folder and click **OK**.

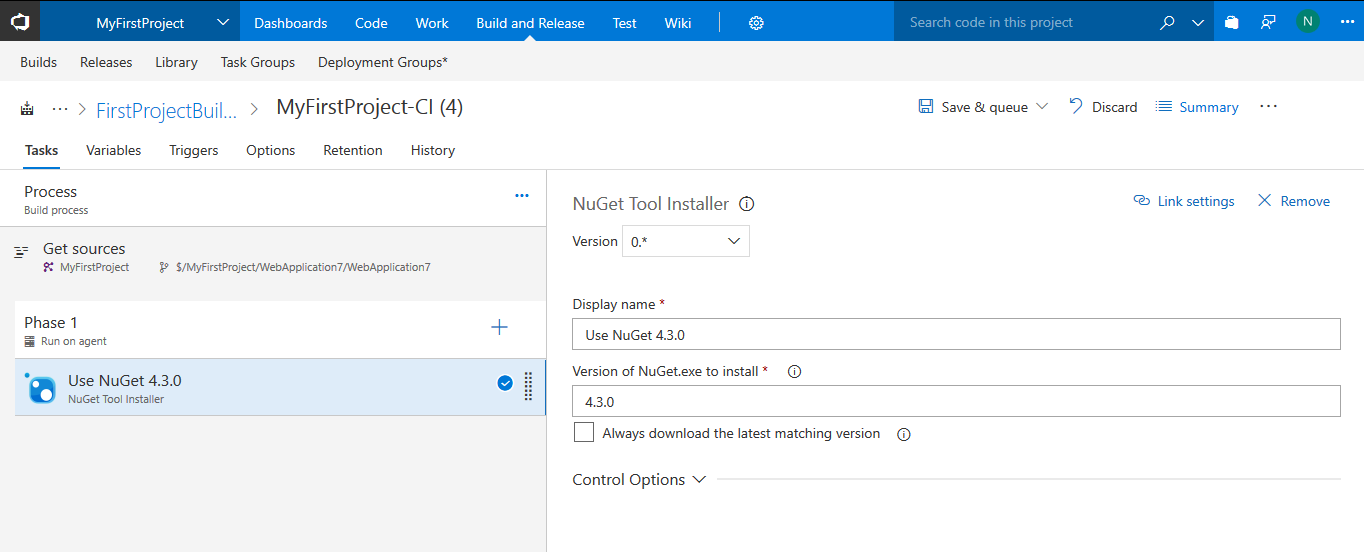


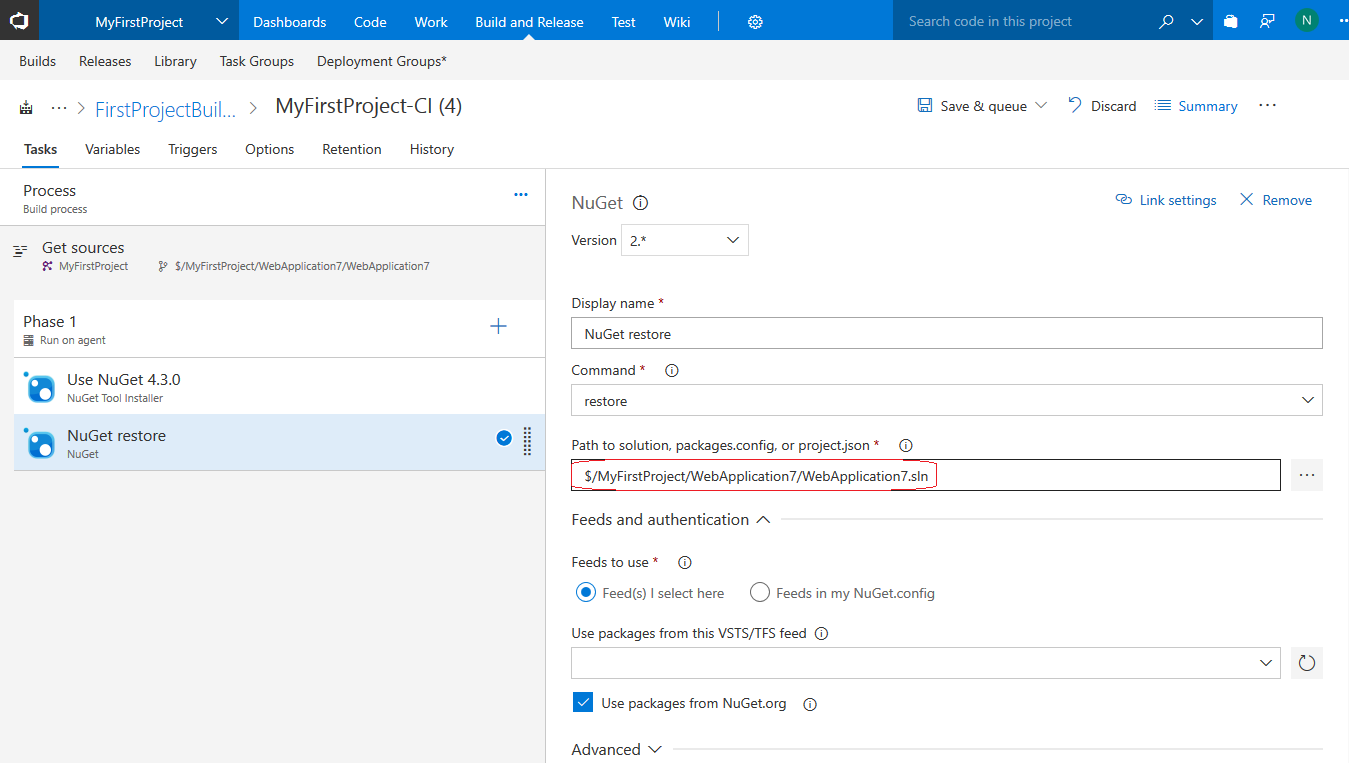


Click on Save. Click on Add **+** next to Phase 1. The tasks that you will add here will ensure that they run on the agent identified.

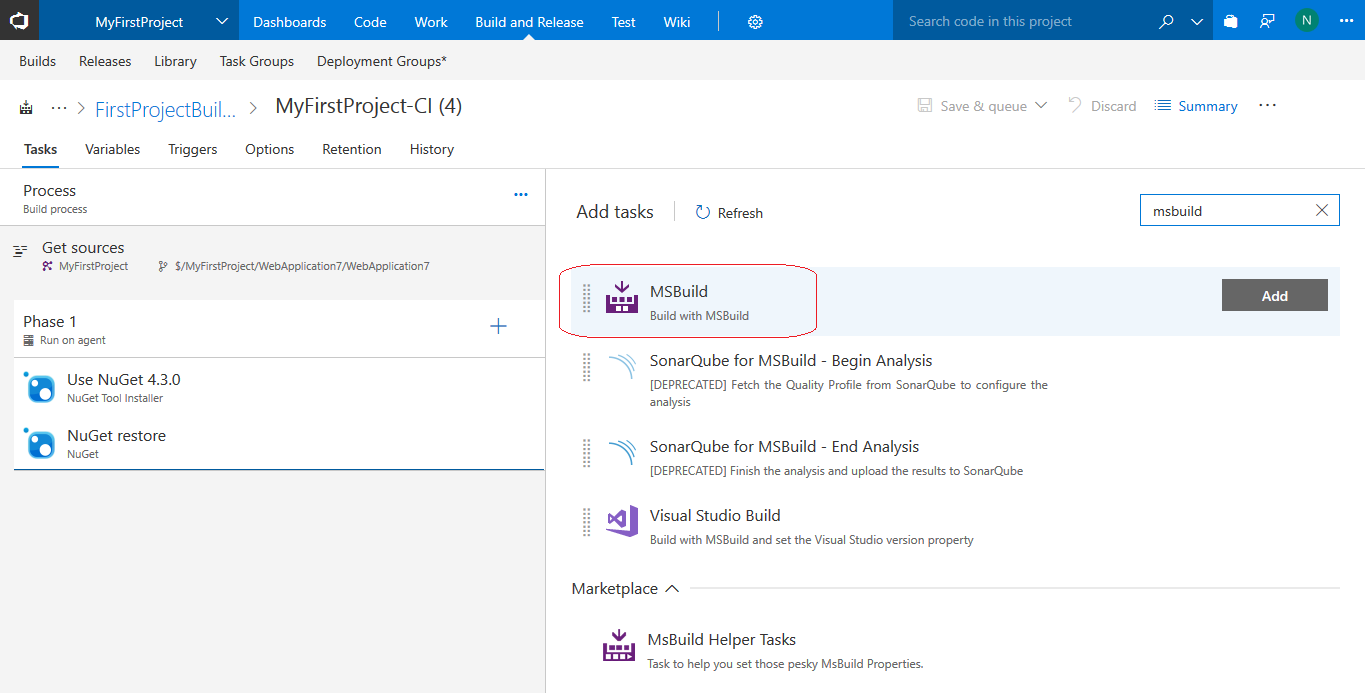
Add two NuGet tasks in the following order to restore the NuGet packages. **First add NuGet Tool Installer and then NuGet Restore.** NuGet is basically designed to share code in .NET with help of packages. More on NuGet can be found @ <https://docs.microsoft.com/en-us/nuget/what-is-nuget>



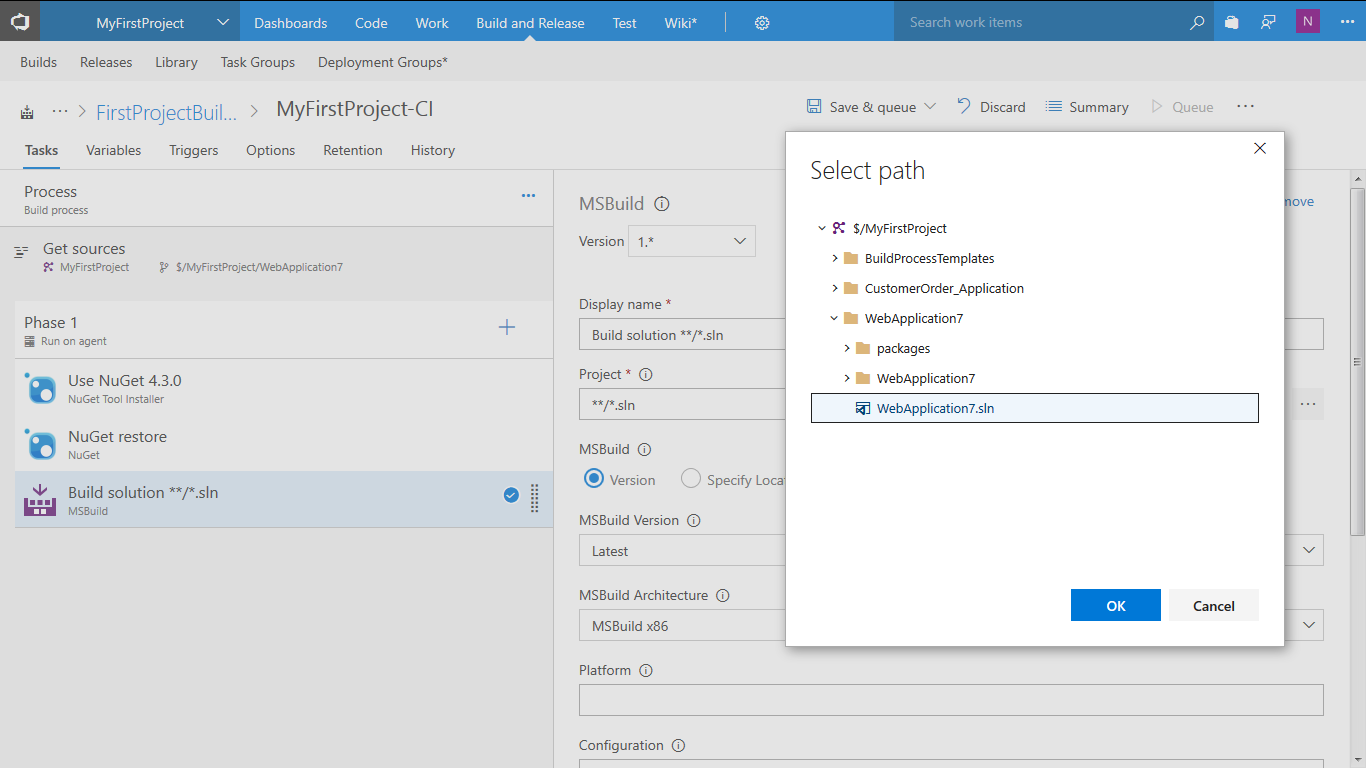


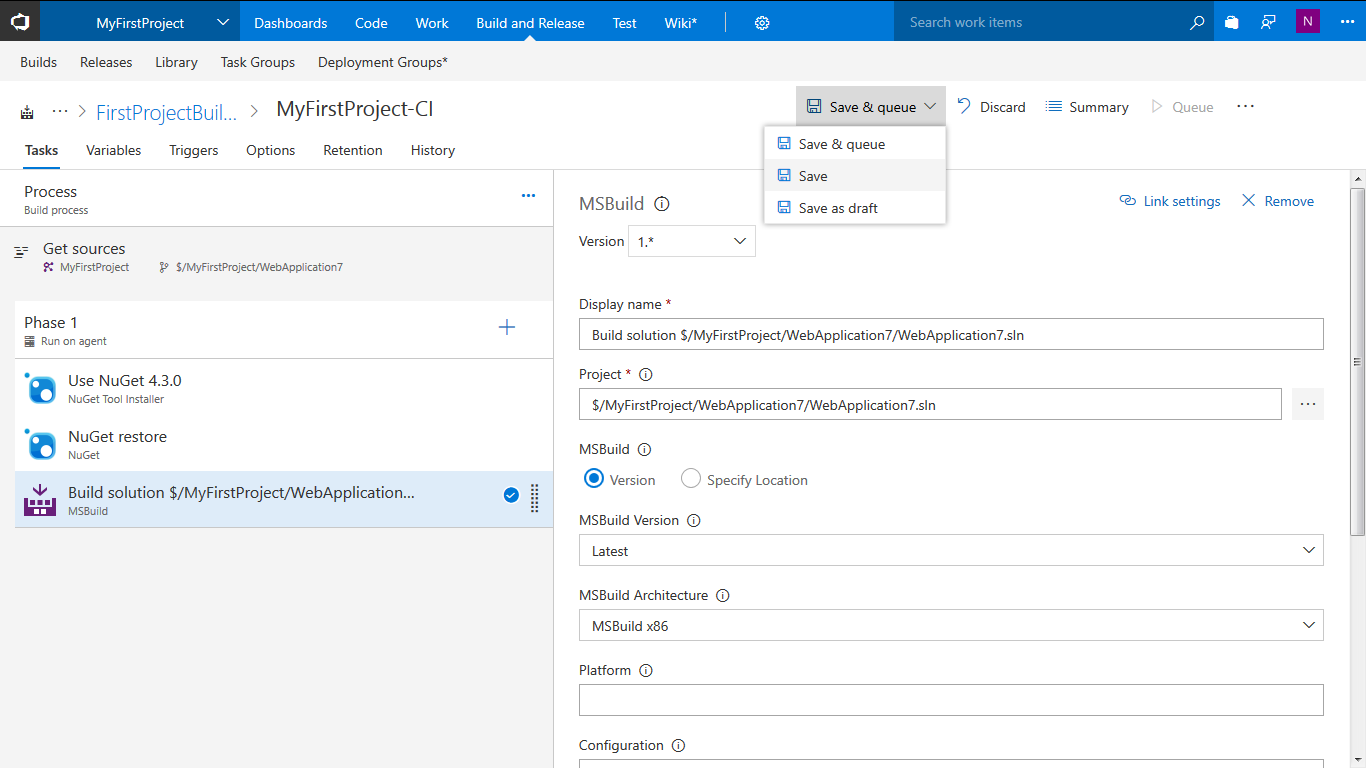


Next under **Build** category **select MS Build** and click on **ADD**



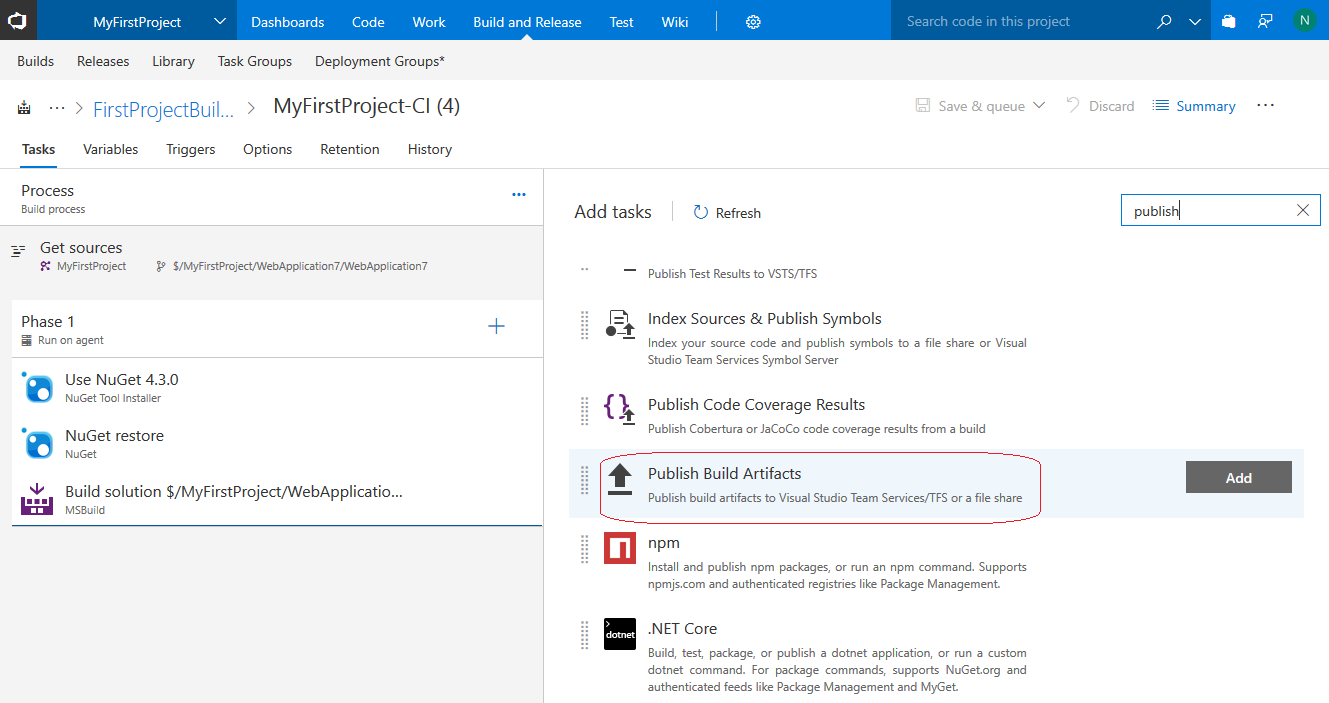
Select the **Build Solution** Step and click on 3 DOTS (…) next to Project box and select the solution file.

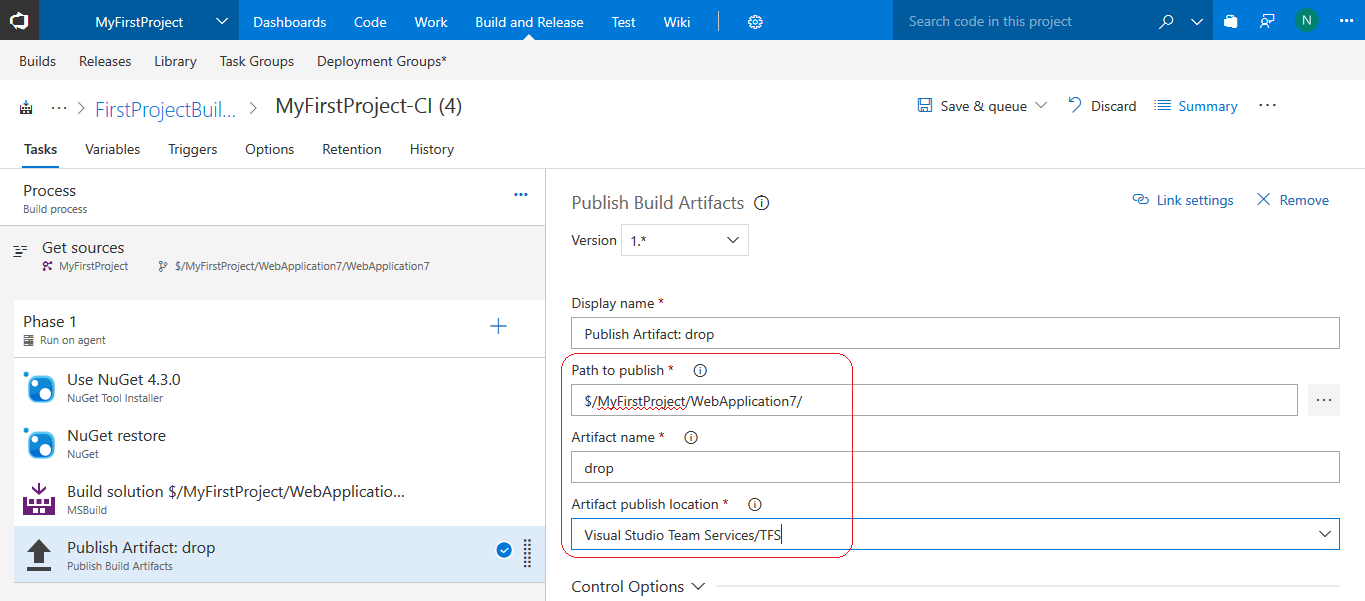




**Save** the Build Definition. Add another step to **Publish** the artifacts to server as a drop folder which will be used later during the Release process.

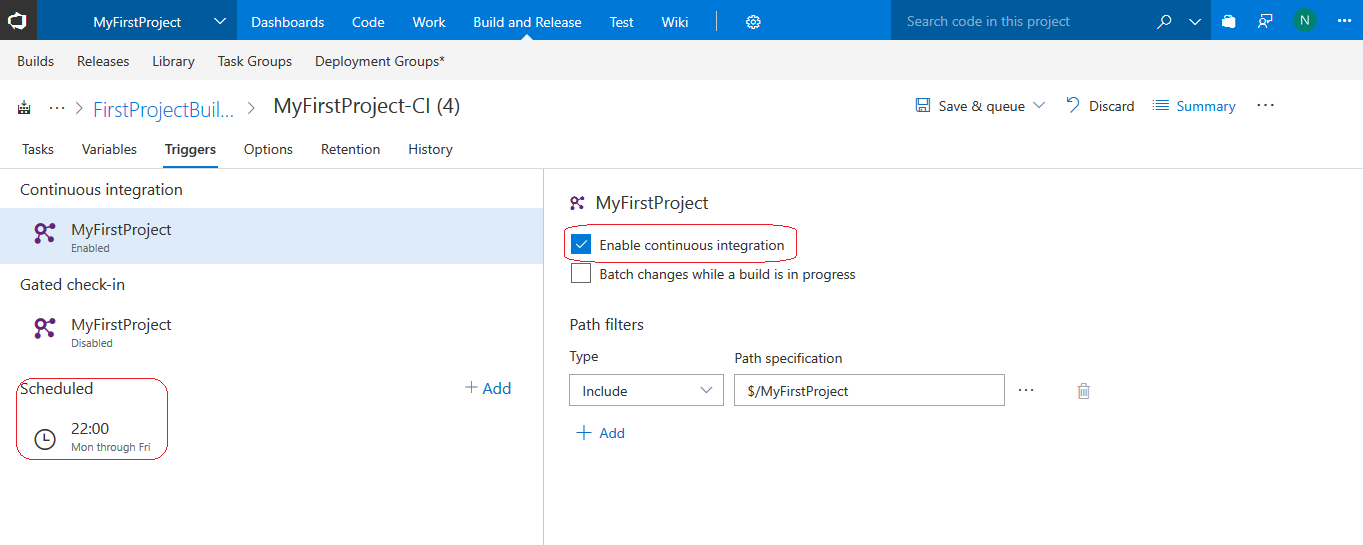
Under **Utility** select the **Publish Build Artifacts** step



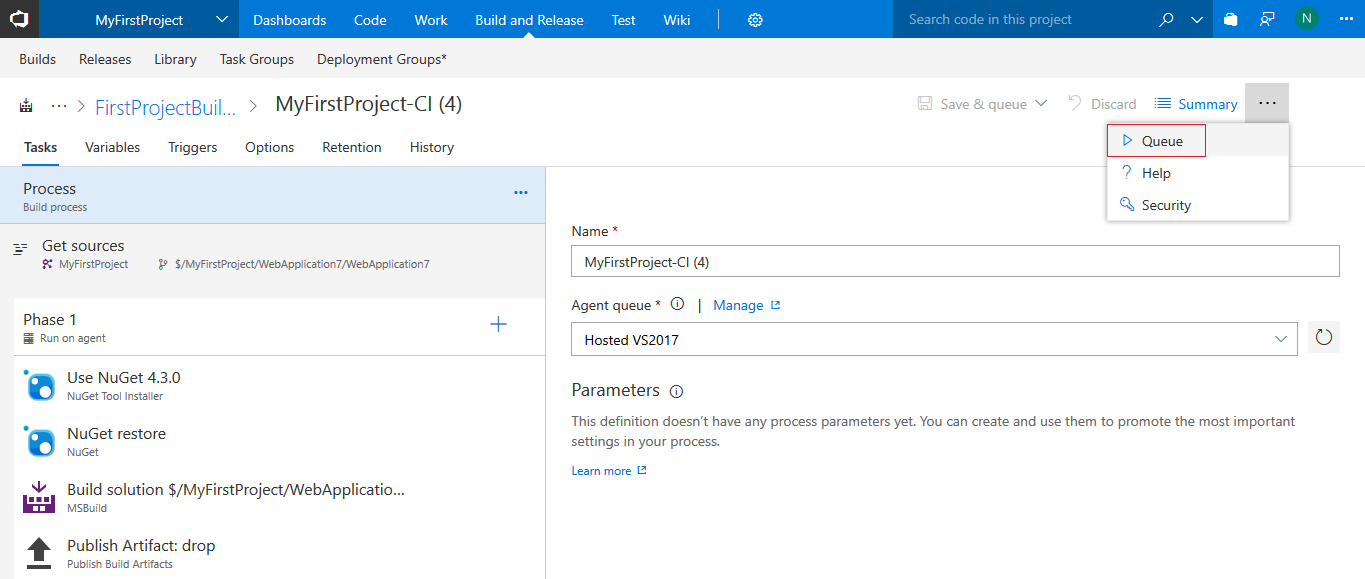


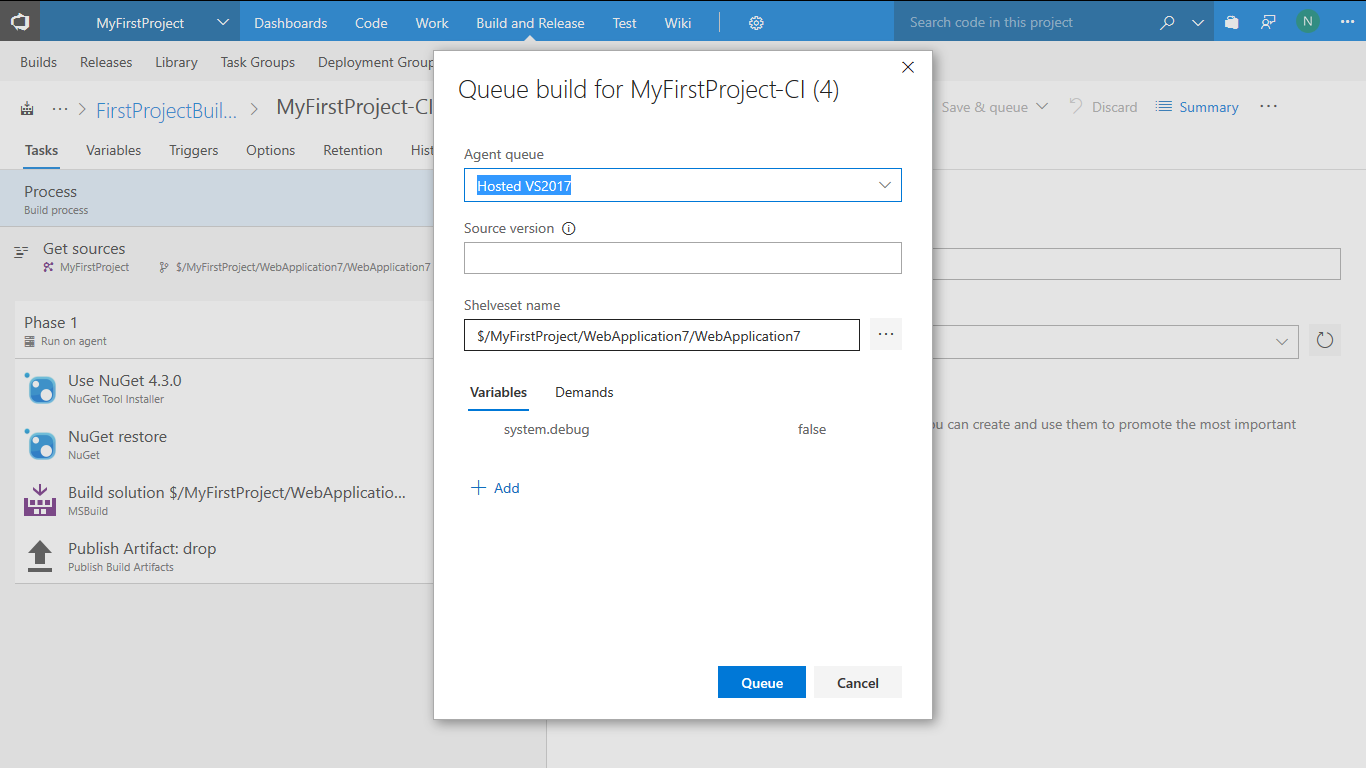
**Save** the build definition.

**Option -** Before triggering a build if needed enable the option to build on every check in in the Triggers TAB of the Build Definition. If not based on the Continuous Integration option then enable the Scheduled option on the right to build the solution during specified week day and time. This will depend on how you want your project to be built.

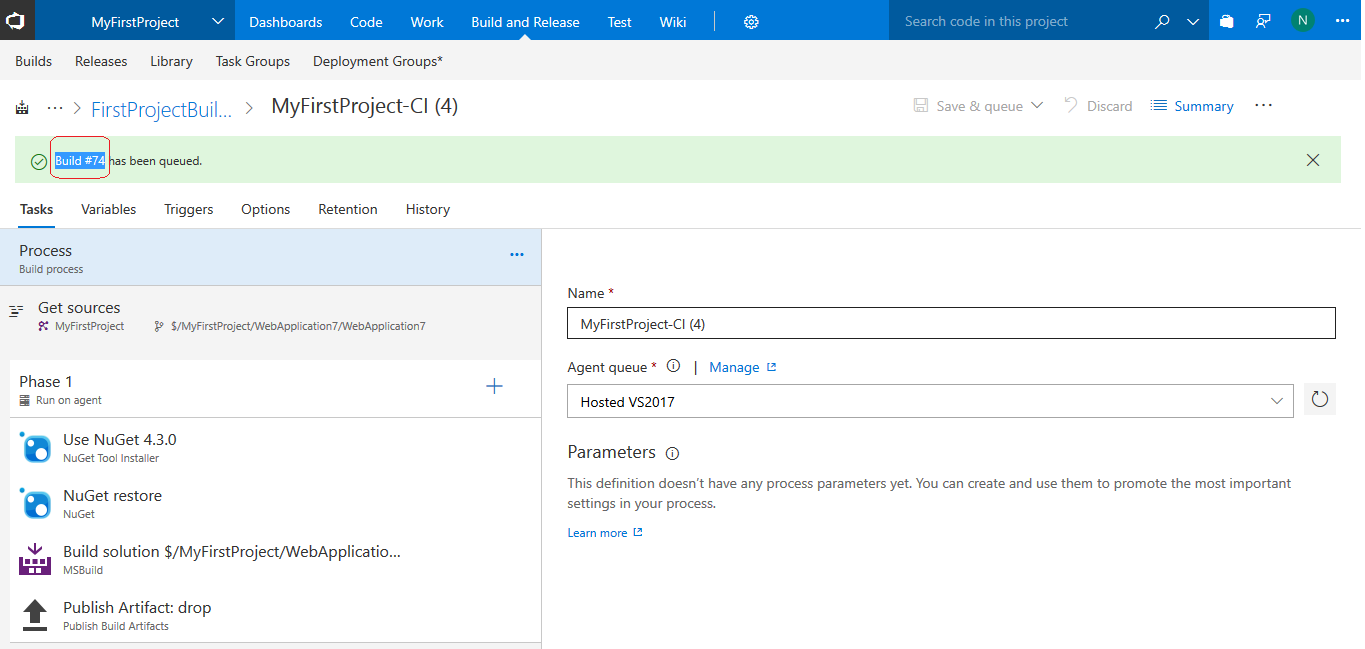


Trigger a build by selecting the Queue option

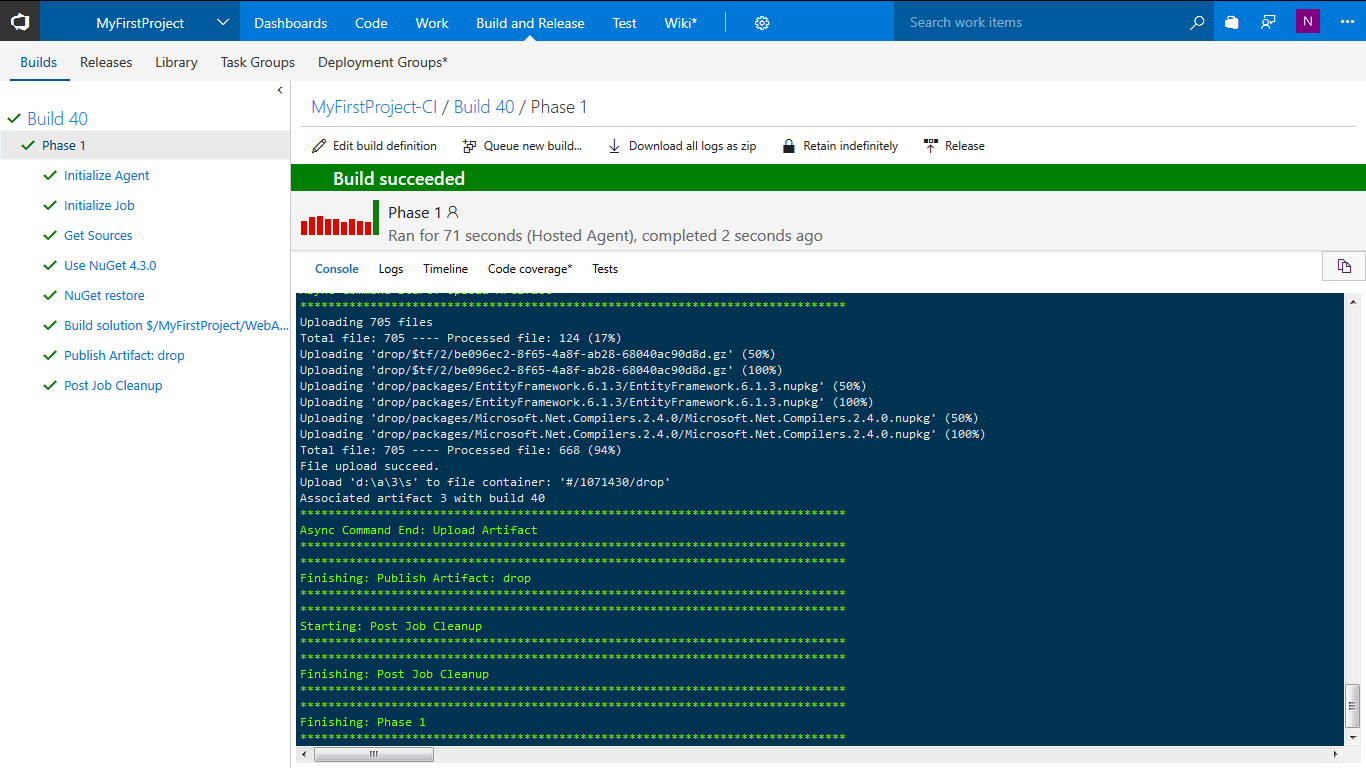




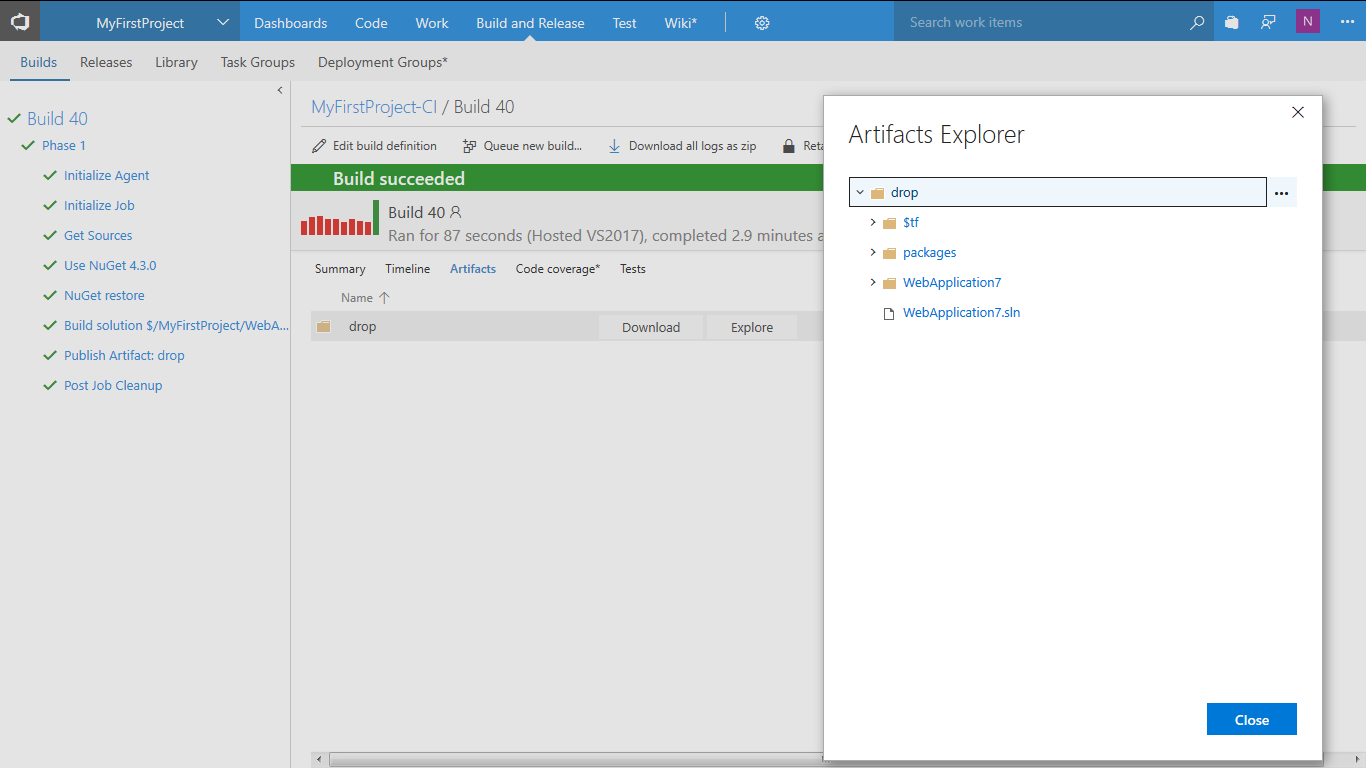
Click on Queue button. Once the build starts click on Build number, as highlighted below, to view the build status.



Build is successful as shown using the inbuilt hosted agent.



Click on the build number (e.g. Build 40 in this screenshot) to verify whether the artifacts have been uploaded to **drop** folder in the server. Click on **Artifacts TAB 🡪 drop folder**.



# 

# Create an Web App resource in the Azure portal

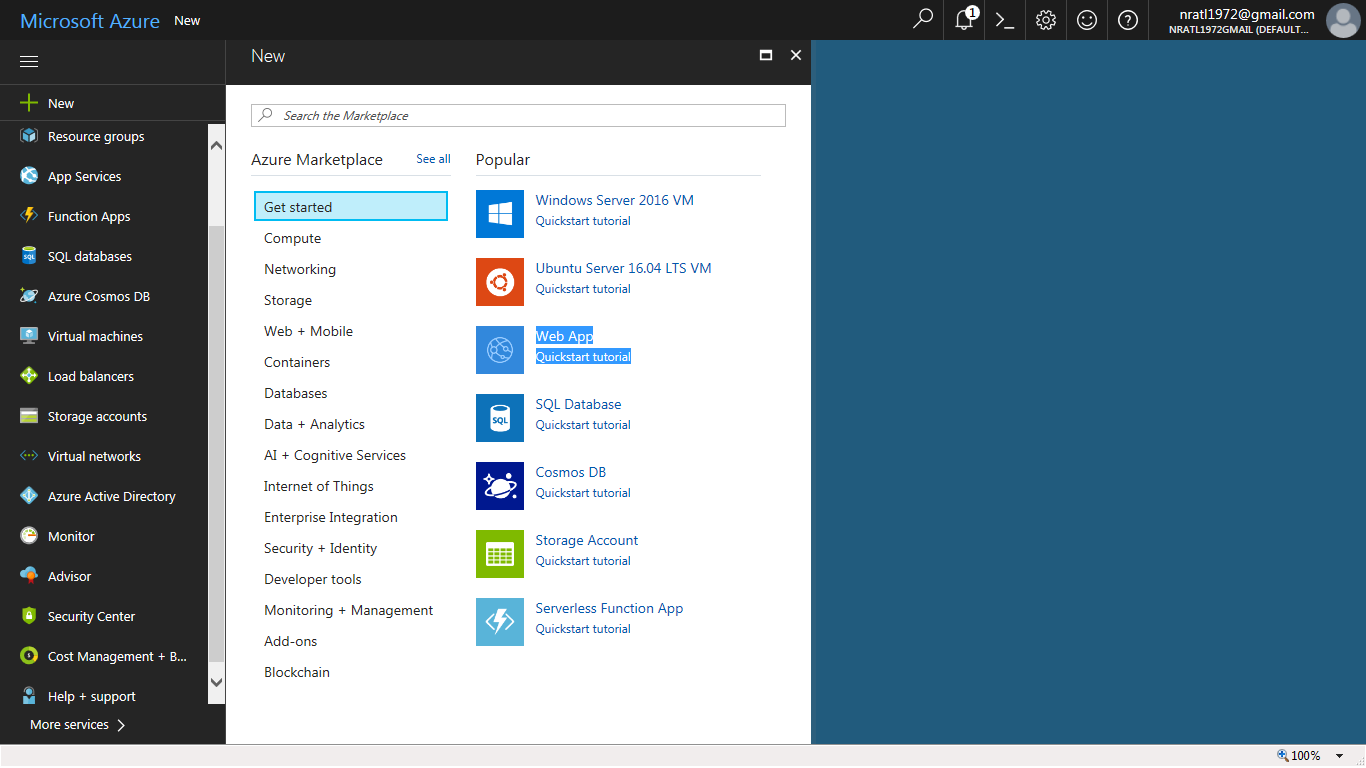
As of the previous section we have defined the Build Definition (Continuous Integration). In this section we will see now see how to set up a Release Definition (Continuous Delivery) process to Azure of the ASP.NET application built earlier.

**Pre-requisite**

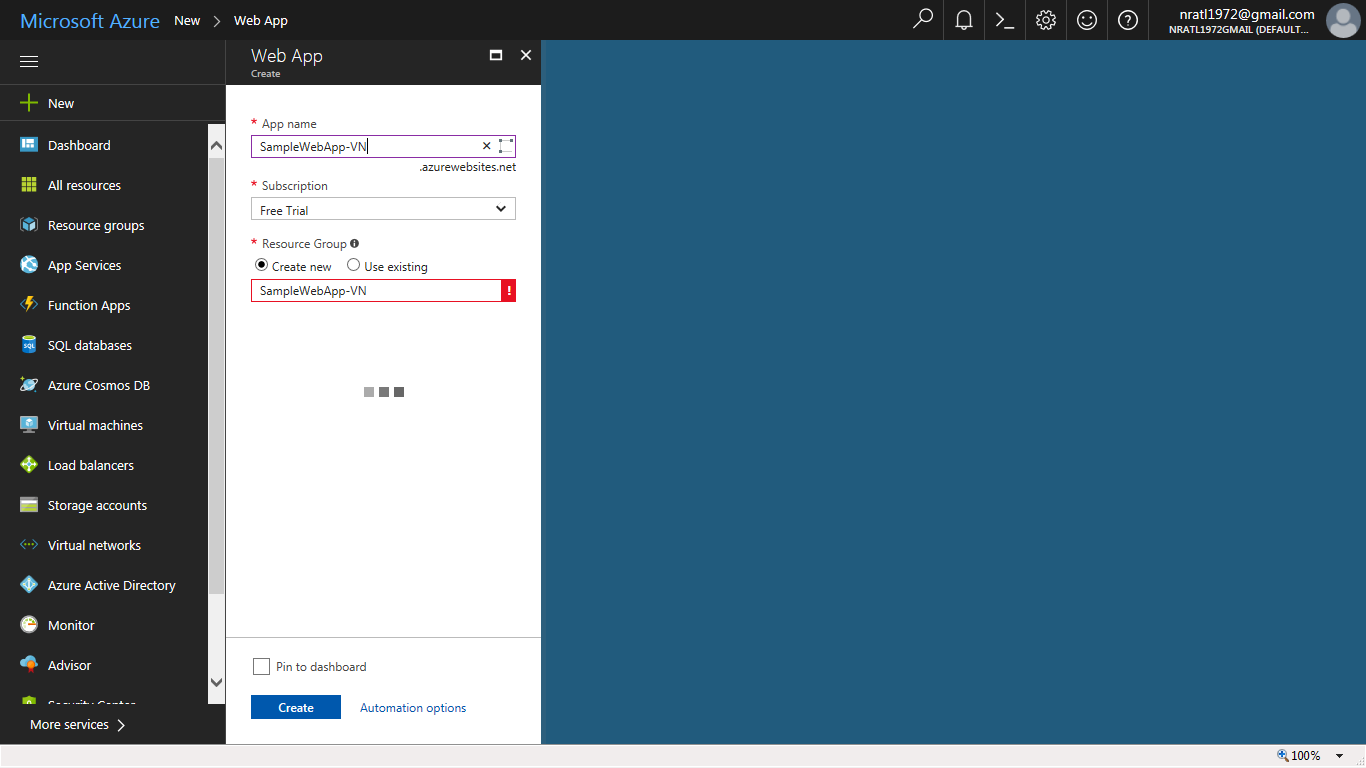
We will have to create a web app using the portal as shown below before proceeding with deploy to Azure Cloud.

**Create an Azure web app using the portal**

1. Sign into the [Microsoft Azure portal](https://portal.azure.com/) which is <https://portal.azure.com>
2. Choose the **+ New** icon in the left navigation bar, then choose **Web App**

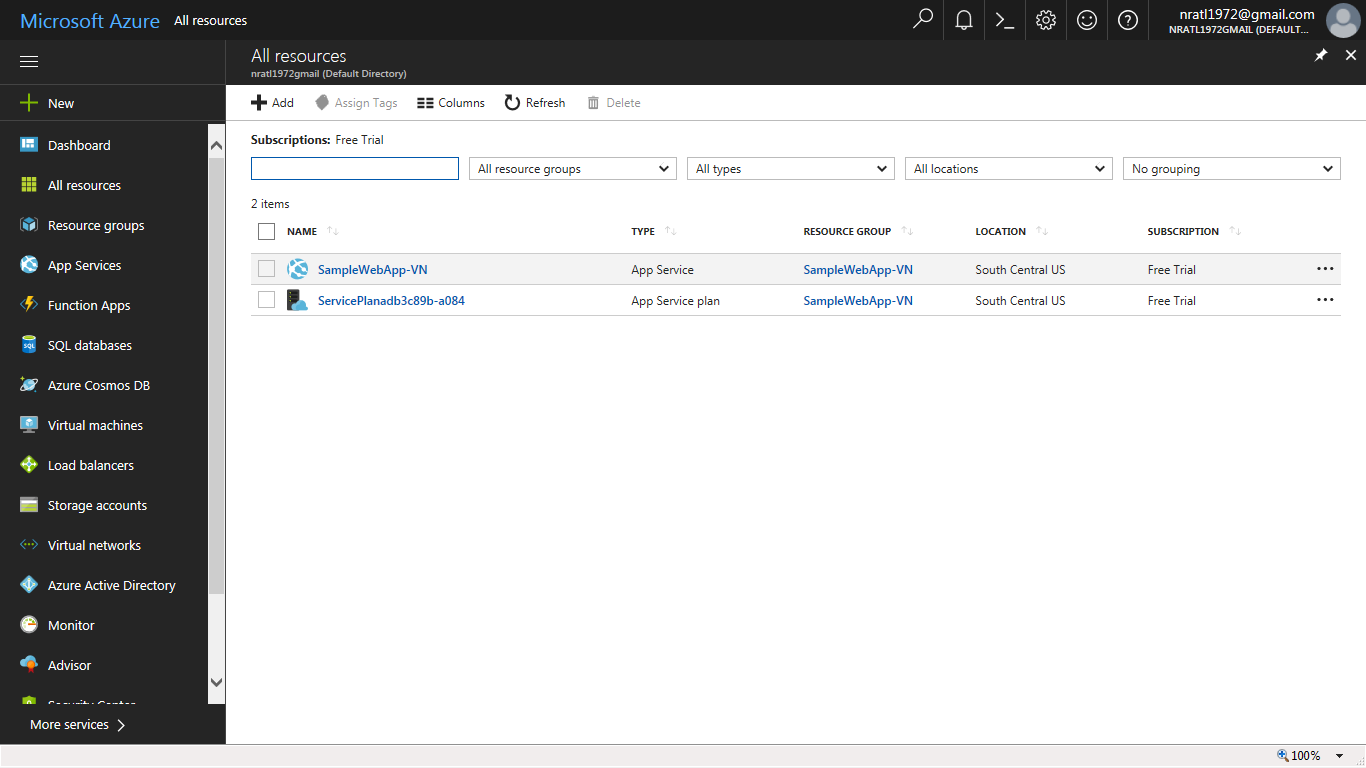


Enter a name for the App which will be used later in the release process to be created in VSTS

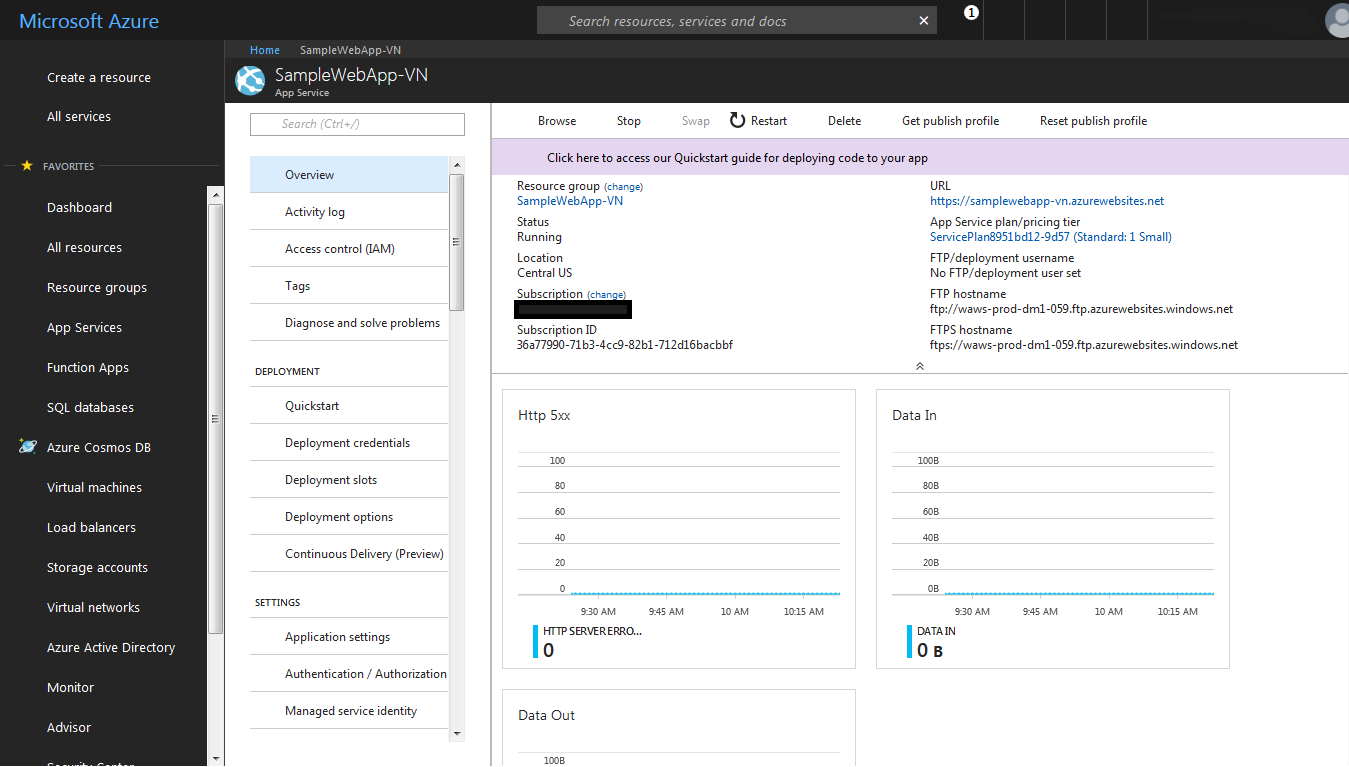


Click **Create**.

**Click on All Resources** to view the web app created.



Click on the resource **SampleWebApp-VN** will display the URL of the application which will be deployed using release process in VSTS

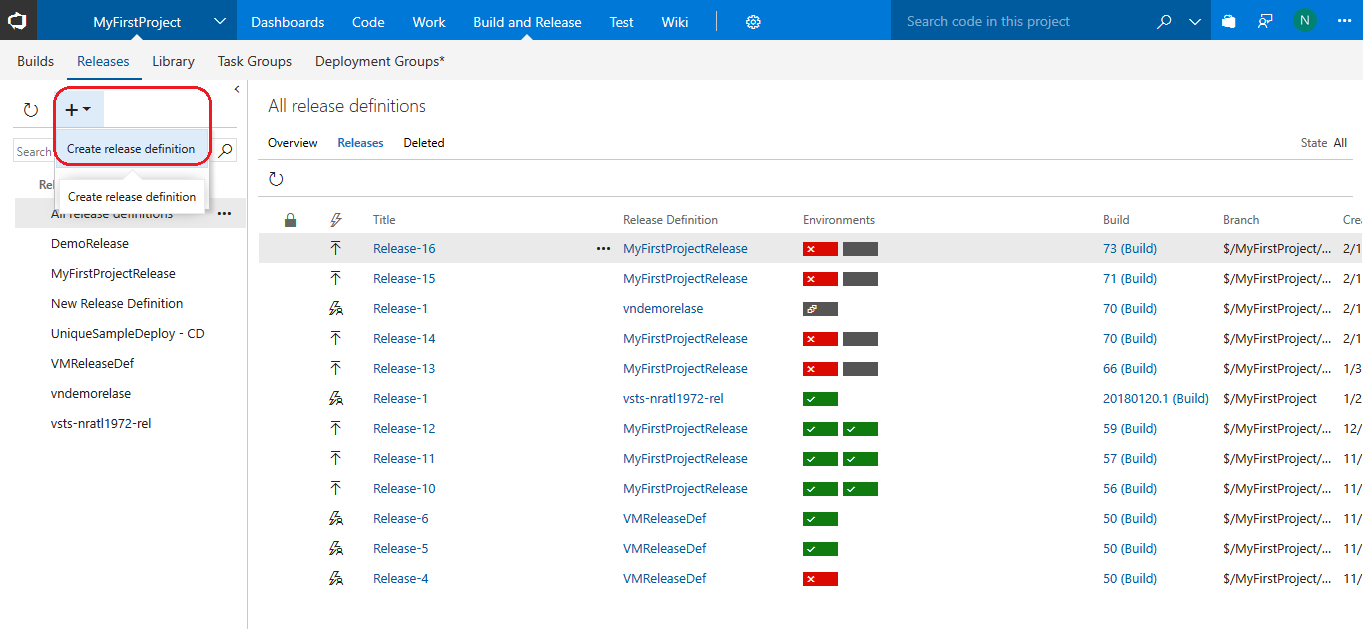
****

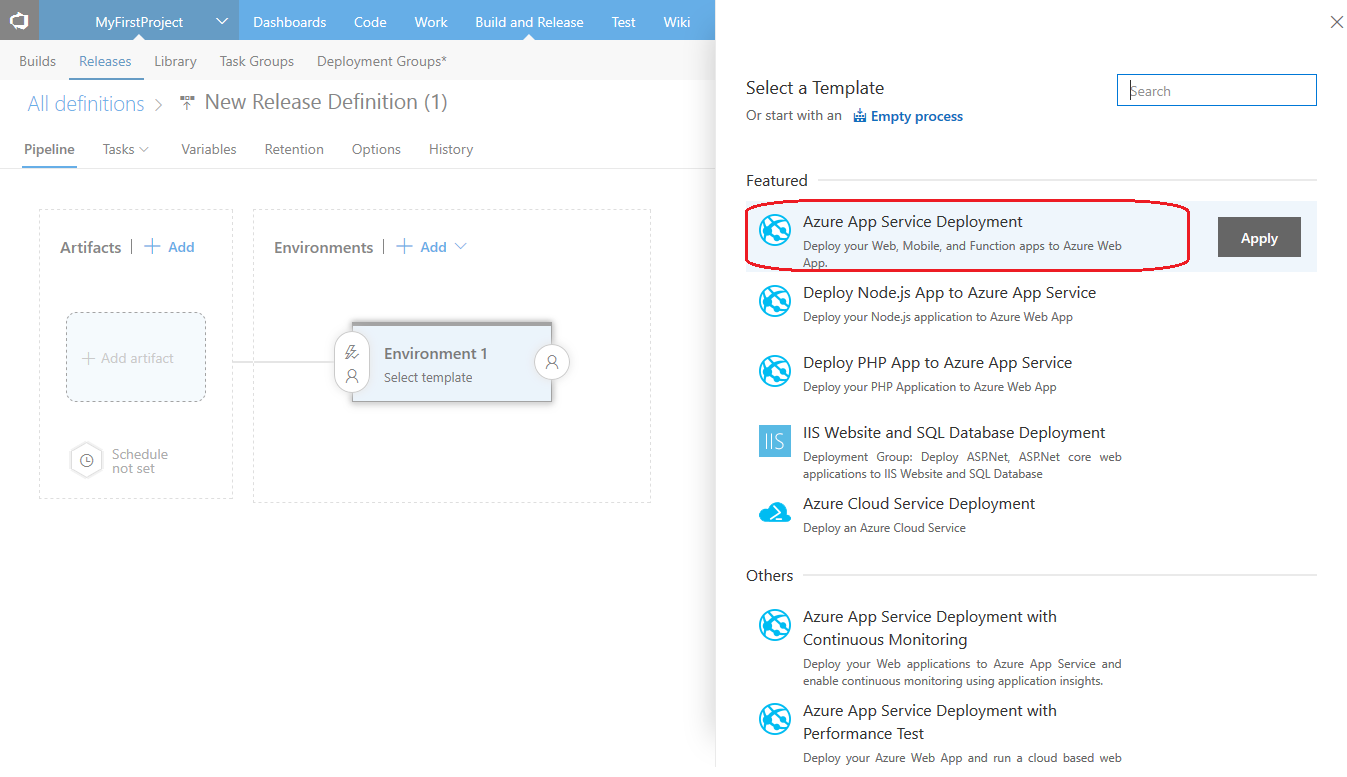
# Create Release Definition (Continuous Delivery)

Since the build artifacts are available in the drop folder as in the build definition the release definition which we will create now for deployment will pick up the artifacts and then deploy them as an Azure web application.

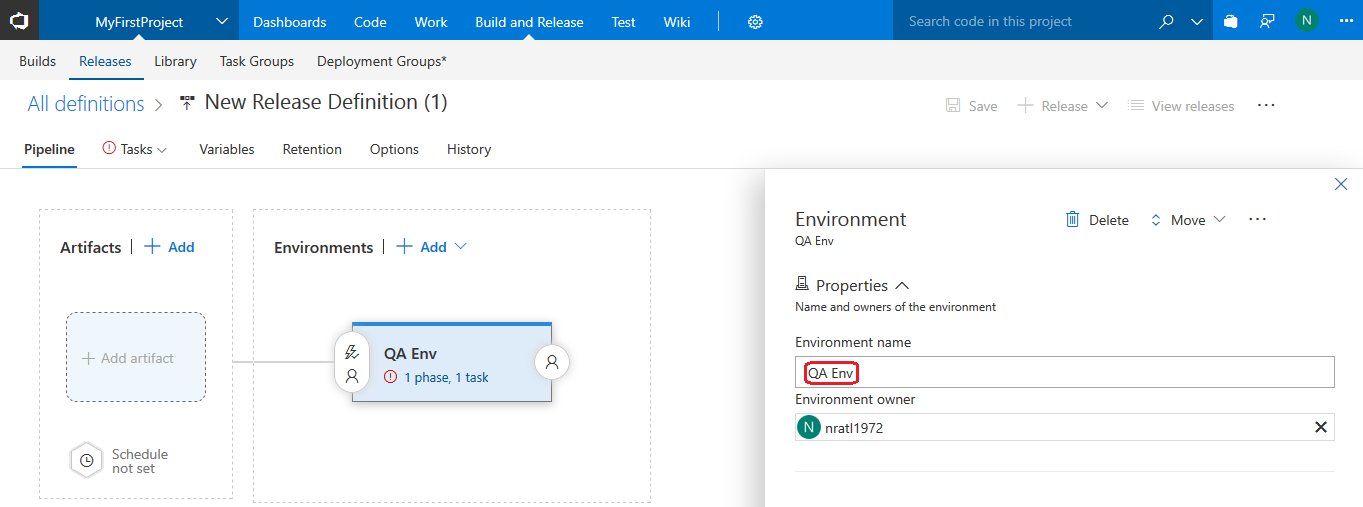
Back in VSTS dashboard create a release definition and link to the build definition which will enable the continuous delivery process

In the **Build and Release** hub, Click and select **Create release definition** as shown.

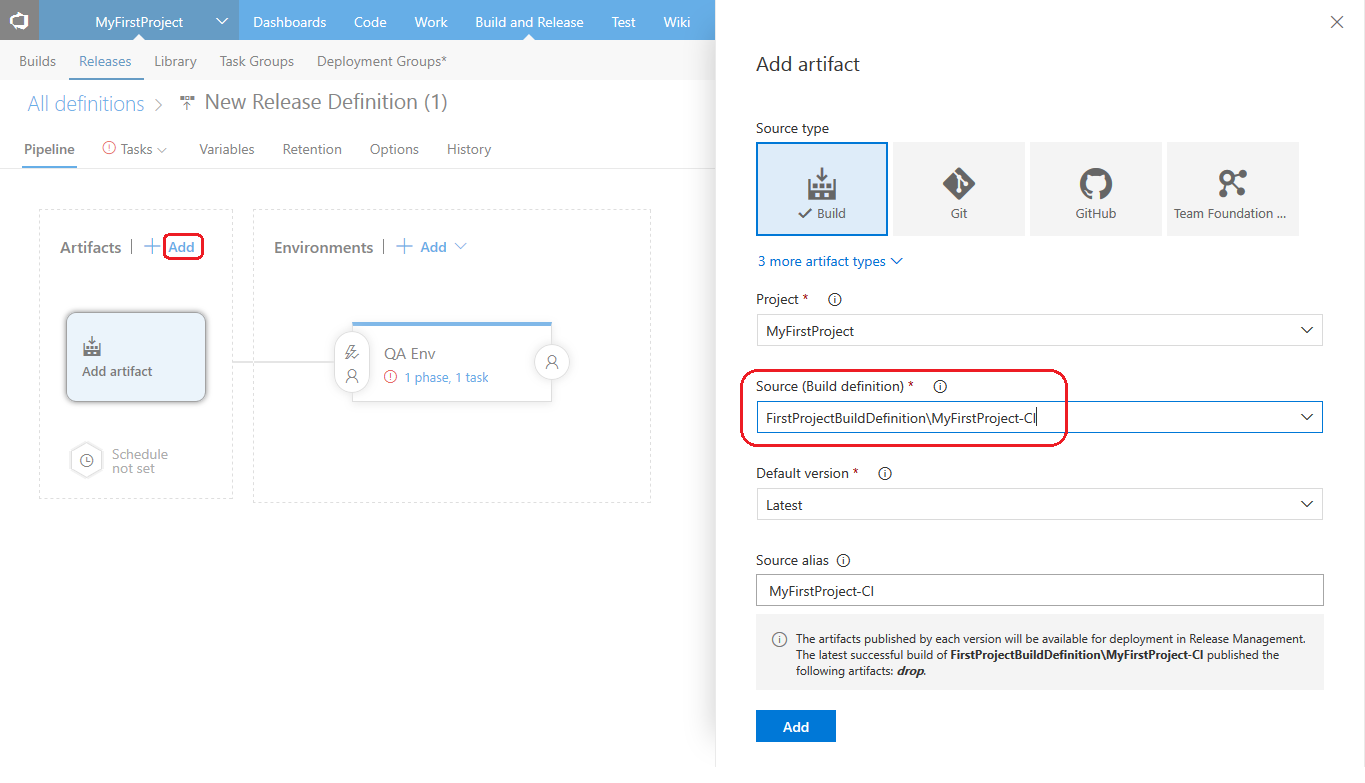


Select **Azure App Service Deployment option** under Featured templates and click on **Apply **

Rename the environment to QA Env

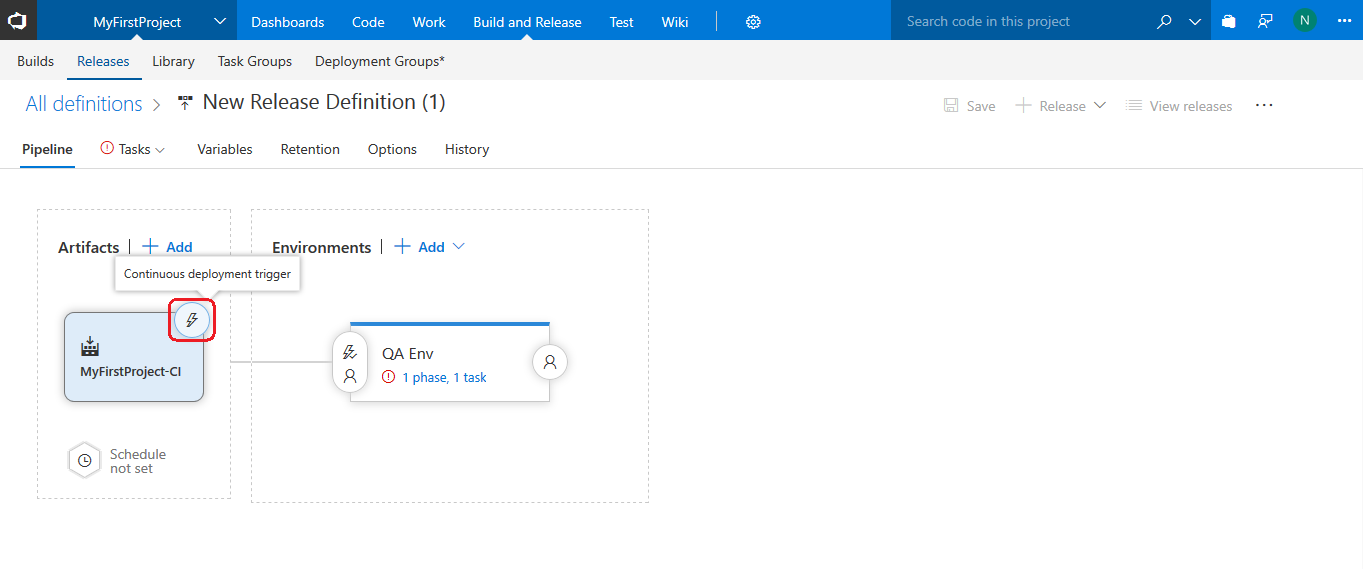


Click on the **+ Add** link under **Artifacts** and select your build definition to link to. Under **Source type** on the rightselect the build definition created earlier.

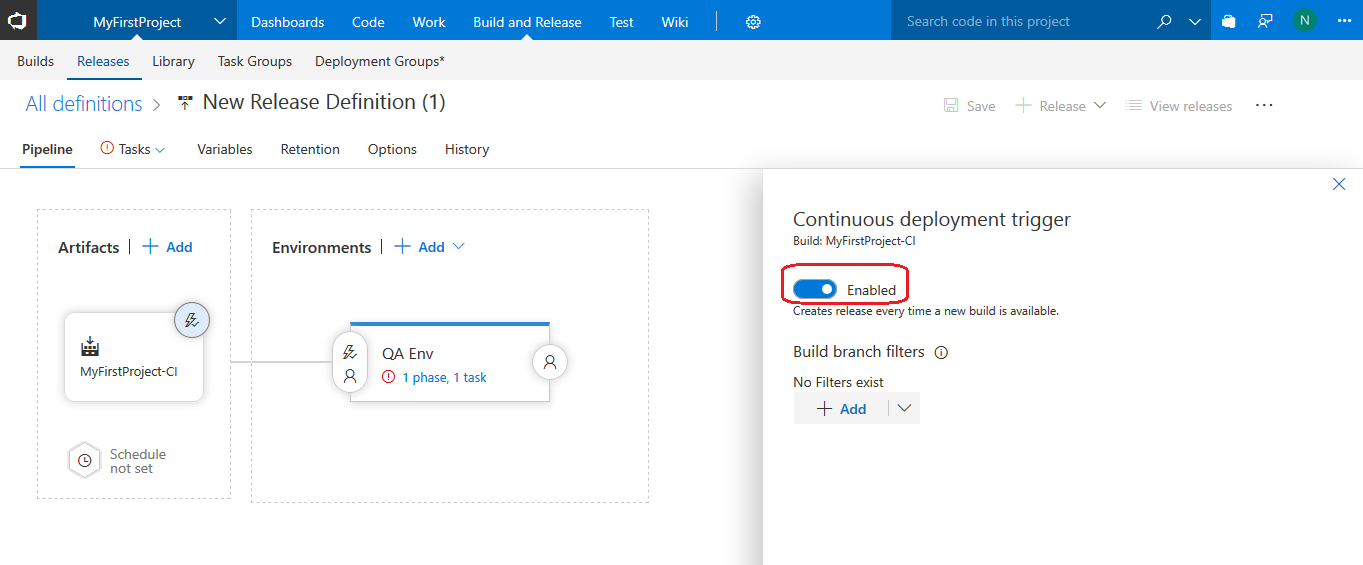


Select the Build Definition created earlier and click on **ADD**

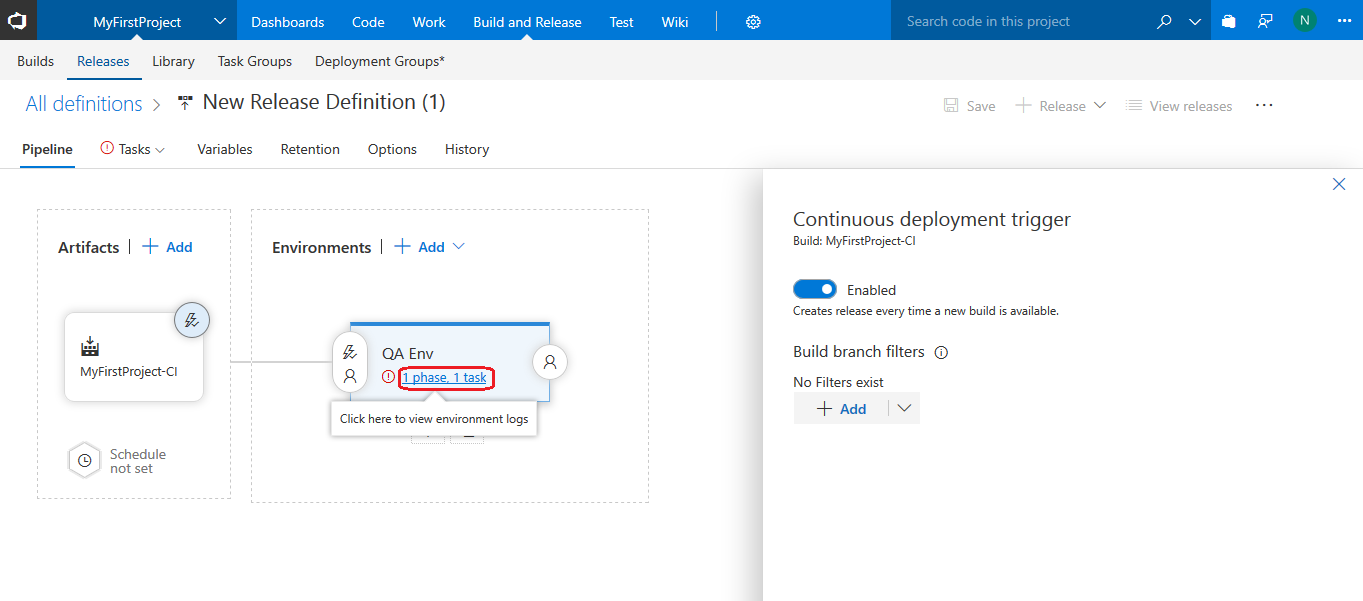
Click on **Continuous Deployment** trigger icon and enable the same. This is to ensure that the deployment starts once the Build is completed.



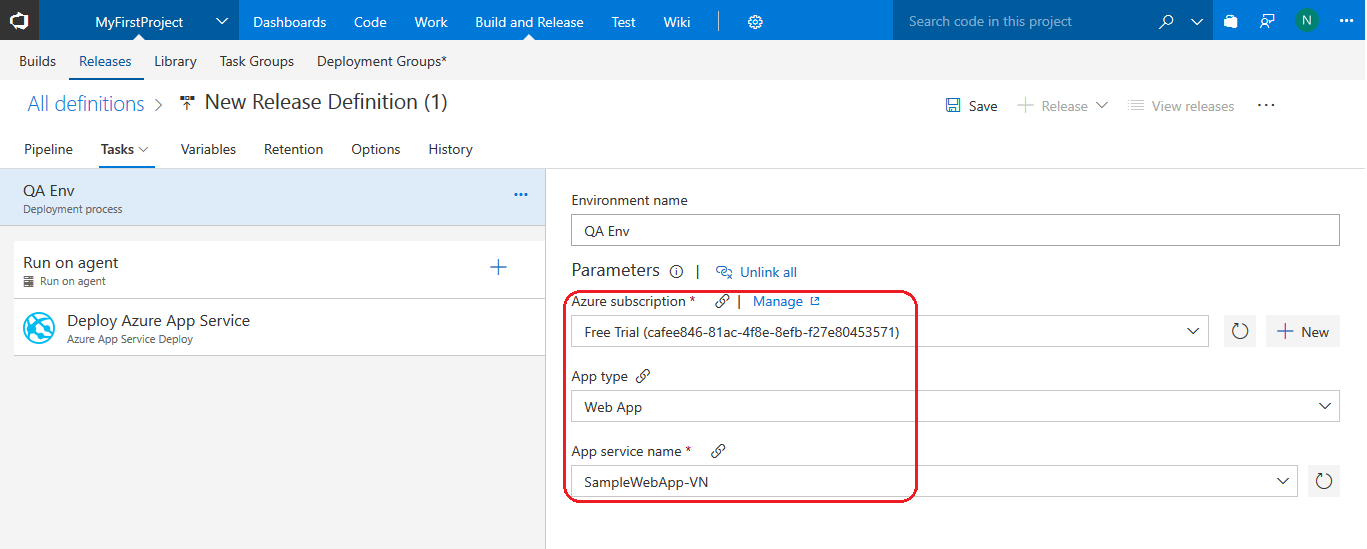
Enable the slider for Continuous deployment trigger. If enabled this ensures the release will start automatically and deployment will be triggered to Azure.



Next click on the link **view environment logs** under the QA Env to start configuring the tasks to deploy to Azure Web App.



Choose the appropriate Azure Subscription and App Service name created in Azure portal earlier after Authorization. Also ensure to select the application to deploy which is in the **drop** folder.

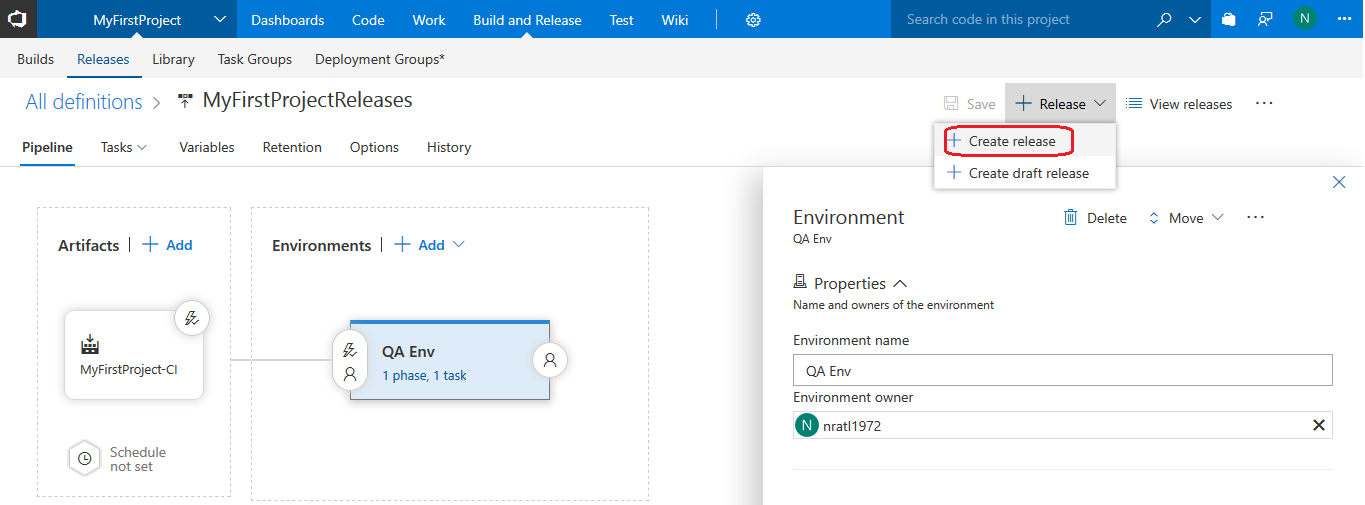


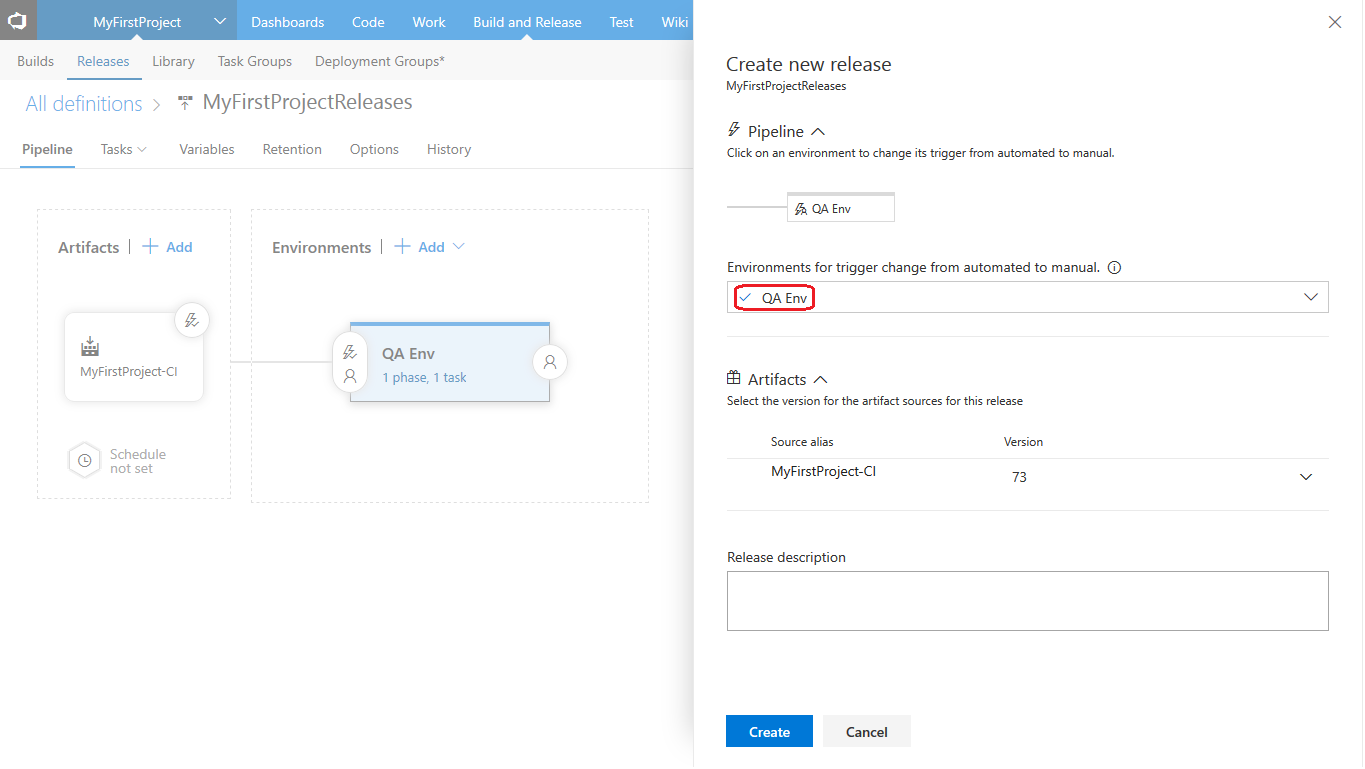
Save the Release Definition.

# Create a release to deploy your app

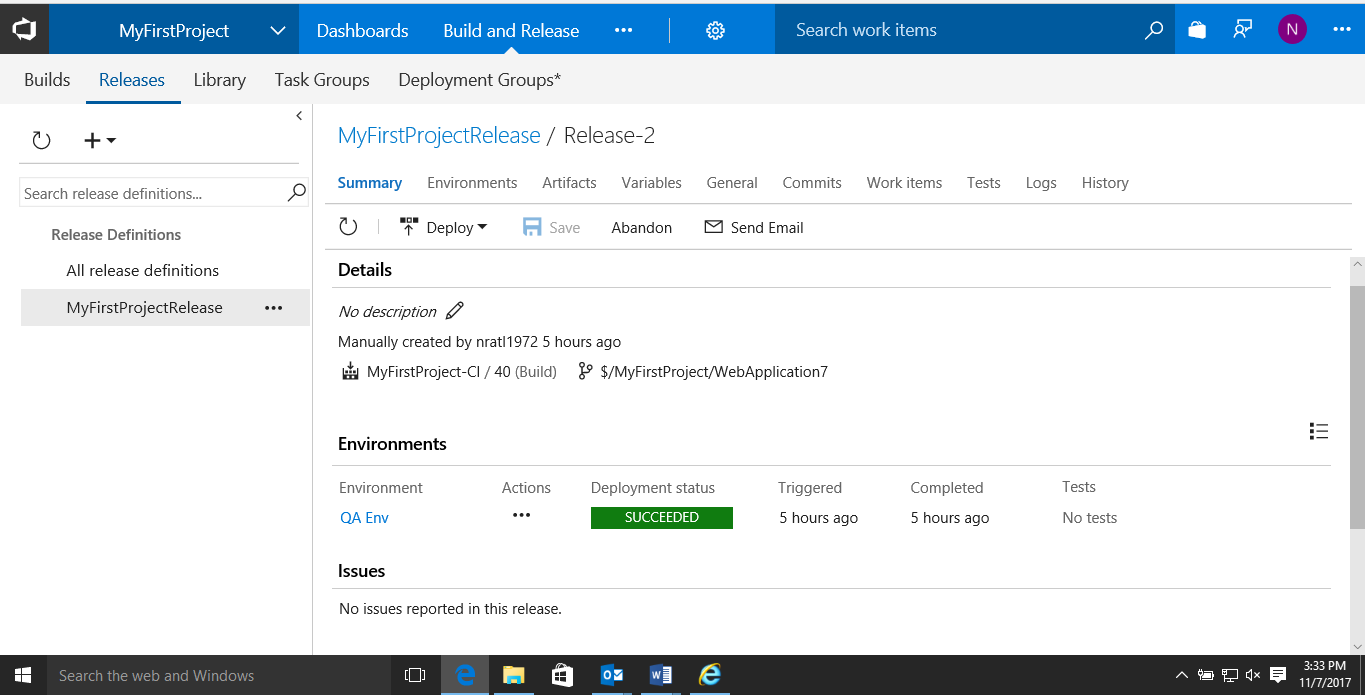
We can now start the deploy process of the build by creating a Release

1. Select **+ Release** drop down and **Create Release**. Select the Environments as QA Env and proceed. The release is now shown as successful in **GREEN TICK**

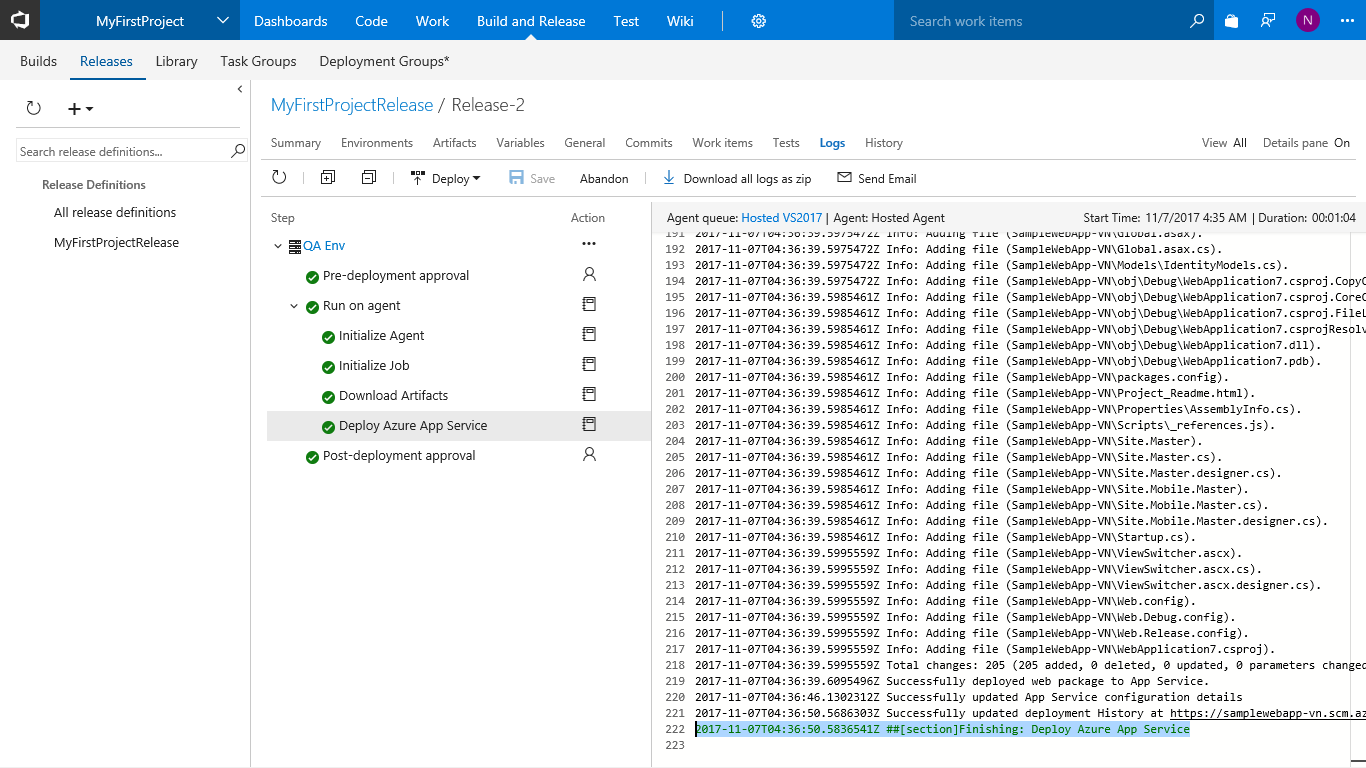




Double click on the **Release-<Number>** under **Title** to look at the Deployment status of the application to all the different environments defined in the release definition.



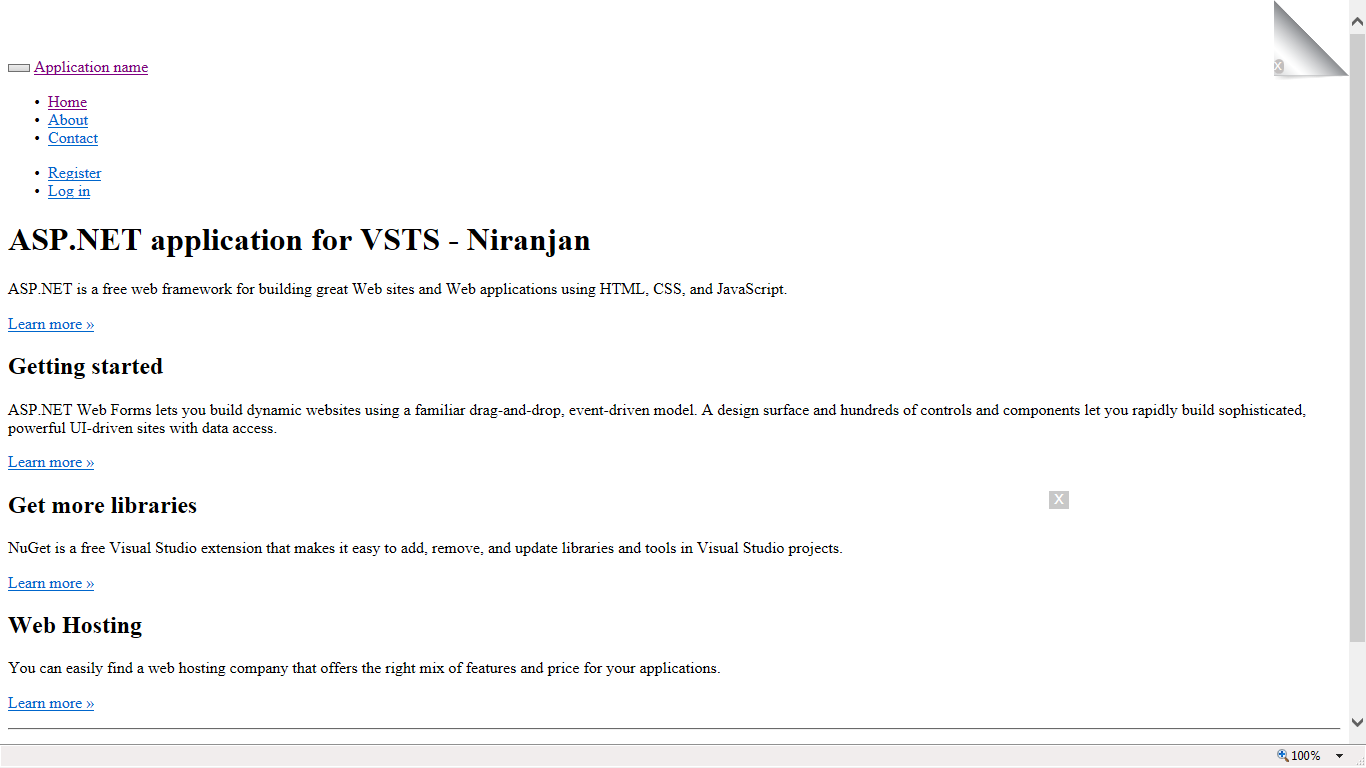
1. The **Logs** tab will provide details of the entire release process steps to all environments run to deploy the web application to Azure.



1. Once the release has run successfully, to view the web application deployed on Azure as a web application browse to the URL provided in the portal

In this example the URL would be <http://samplewebapp-vn.azurewebsites.net/>

The ASP.net web application is now deployed to Azure



# Summary

In this two part series we have seen how Microsoft VSTS can be used as a ALM tool (Work Items, Agile Project Planning, Version Control) on the cloud and can be extended for automating the build (Continuous Integration) and release process (Continuous Delivery) to Azure Web App.

So for Devops enthusiasts and .NET project teams Microsoft VSTS would be an ideal tool for end to end delivery of your software applications.