Lab 02

Build Management using VSTS

Hands on Lab



**Conditions and Terms of Use**

**Microsoft Confidential - For Internal Use Only**

This training package is proprietary and confidential, and is intended only for uses described in the training materials. Content and software is provided to you under a Non-Disclosure Agreement and cannot be distributed. Copying or disclosing all or any portion of the content and/or software included in such packages is strictly prohibited.

The contents of this package are for informational and training purposes only and are provided "as is" without warranty of any kind, whether express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Training package content, including URLs and other Internet Web site references, is subject to change without notice. Because Microsoft must respond to changing market conditions, the content should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

**Copyright and Trademarks**

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

For more information, see Use of Microsoft Copyrighted Content at  
[*http*://www.microsoft.com/about/legal/permissions/](http://www.microsoft.com/about/legal/permissions/)

Microsoft®, Internet Explorer®, and Windows® are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other Microsoft products mentioned herein may be either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are property of their respective owners.

© 2016 Microsoft Corporation. All rights reserved.

Contents

[Introduction 4](#_Toc450038556)

[Exercise 1 – Create a build definition 4](#_Toc450038559)

[Exercise 2 – Restoring solution packages 11](#_Toc450038560)

[Exercise 3 – Protecting the master branch 12](#_Toc450038561)

[Exercise 4 – Using variables in the build definition 16](#_Toc450038562)

[Exercise 5 – Configuring more options: 18](#_Toc450038563)

[Exercise 6 – Executing scripts as part of the build 20](#_Toc450038564)

Exercise 7 – Extending the build tasks from the VSTS Marketplace 25

[Exercise 8 – Cloning the Build Definitions 25](#_Toc450038565)

# Introduction

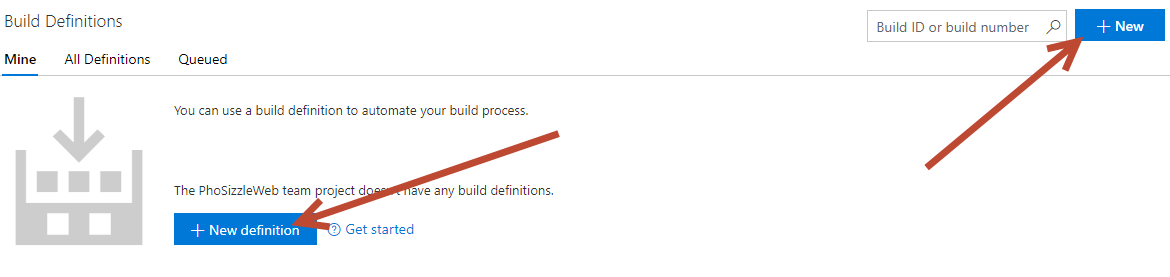
**Duration: 45 minutes**

Goal: Introduction to VSTS Build Management Capabilities

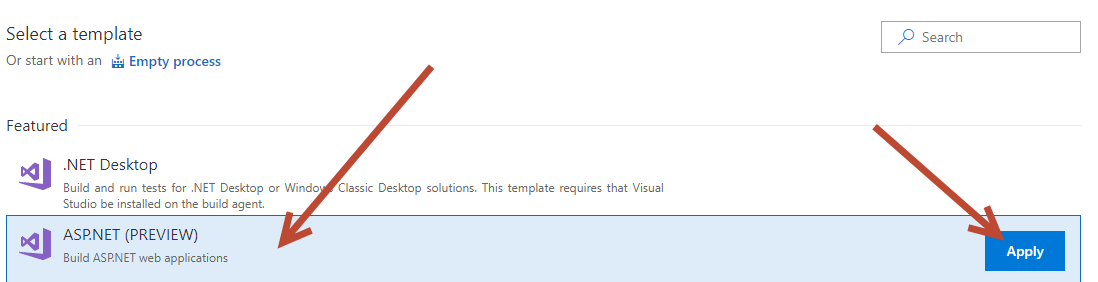
# Exercise 1 – Create a build definition

Our goal is to create a Continuous Integration build that will run anytime some code is pushed into the master branch. We will then enforce that code cannot be merged into master without a Pull request and a successful build.

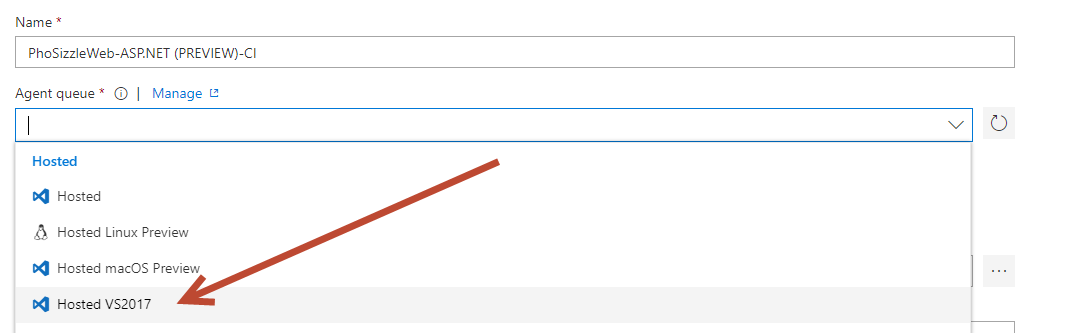
* From the VSTS, click the Build tab (so far there aren’t any build definitions)
* Click on the “Create New Build Definition” button:



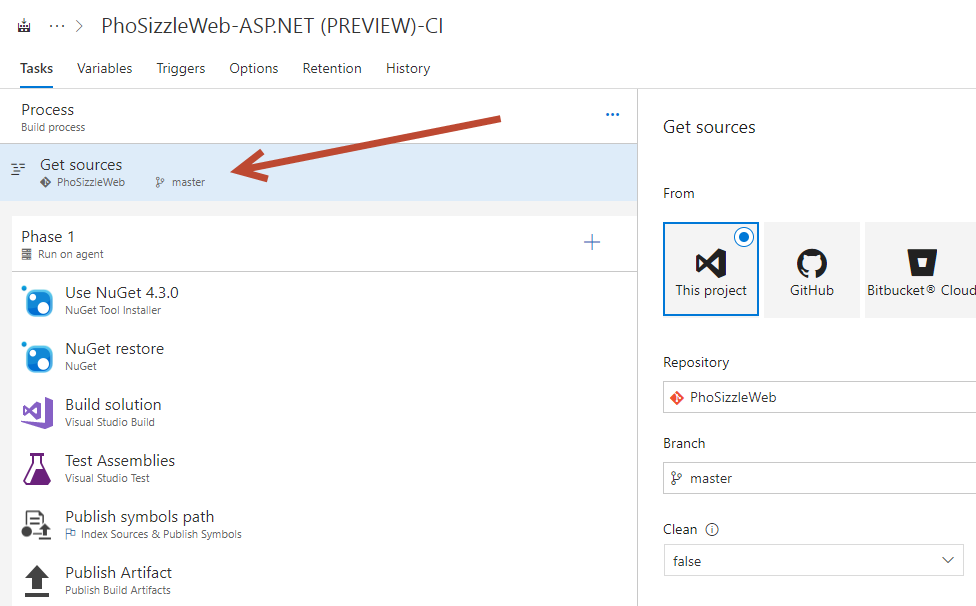
* In the first screen, pick the ASP.NET one (we could have started form an empty template. The ASP.NET template has some tasks that will be needed anyway).
* Click Apply



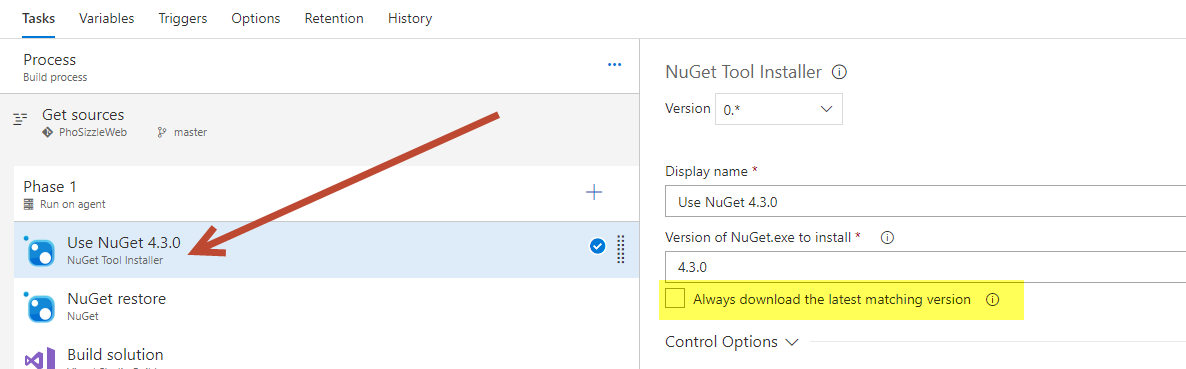
* You will land on the Process tab
* Choose the Agent Queue. Since we are on VSTS we have hosted agents already existing. Choose Hosted VS2017. As a note, the Hosted one is similar but uses VS2015.
  + Please check out this page to see more details about each hosted agent and the software on each one: <https://docs.microsoft.com/en-us/vsts/build-release/concepts/agents/hosted>
  + If you require software not on the hosted agent, require a specific version of software not on the agent, or would like your own VM to be an agent, here more info on setting that up: <https://docs.microsoft.com/en-us/vsts/build-release/concepts/agents/pools-queues>
  + Windows agent deploy documentation: <https://docs.microsoft.com/en-us/vsts/build-release/actions/agents/v2-windows>



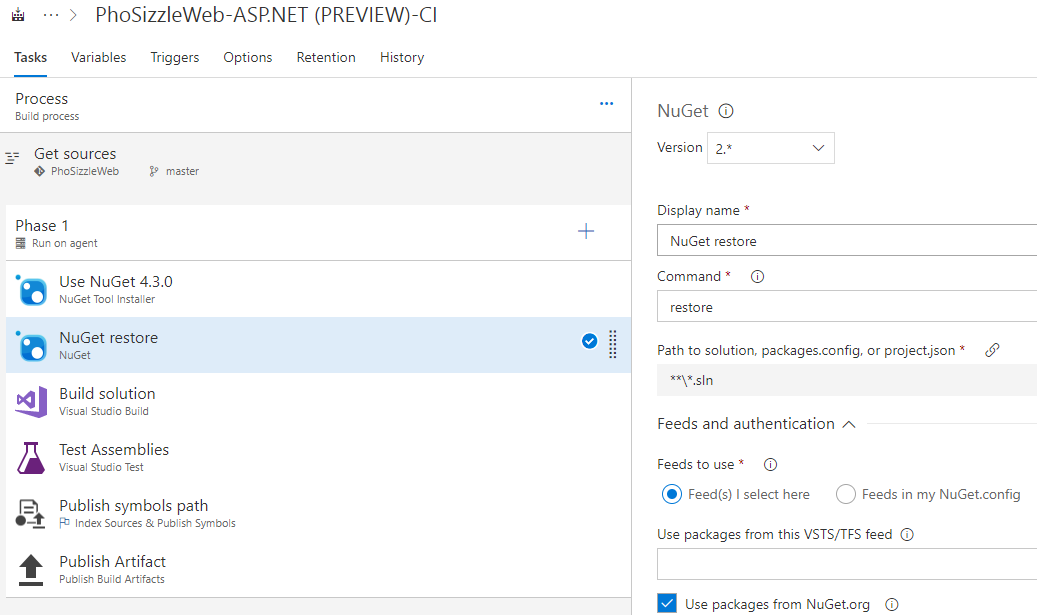
* After choosing Hosted VS2017, click on the next tab down called Get Sources. We don’t need to change anything here, just want to verify we are on the right Repo with the right branch.



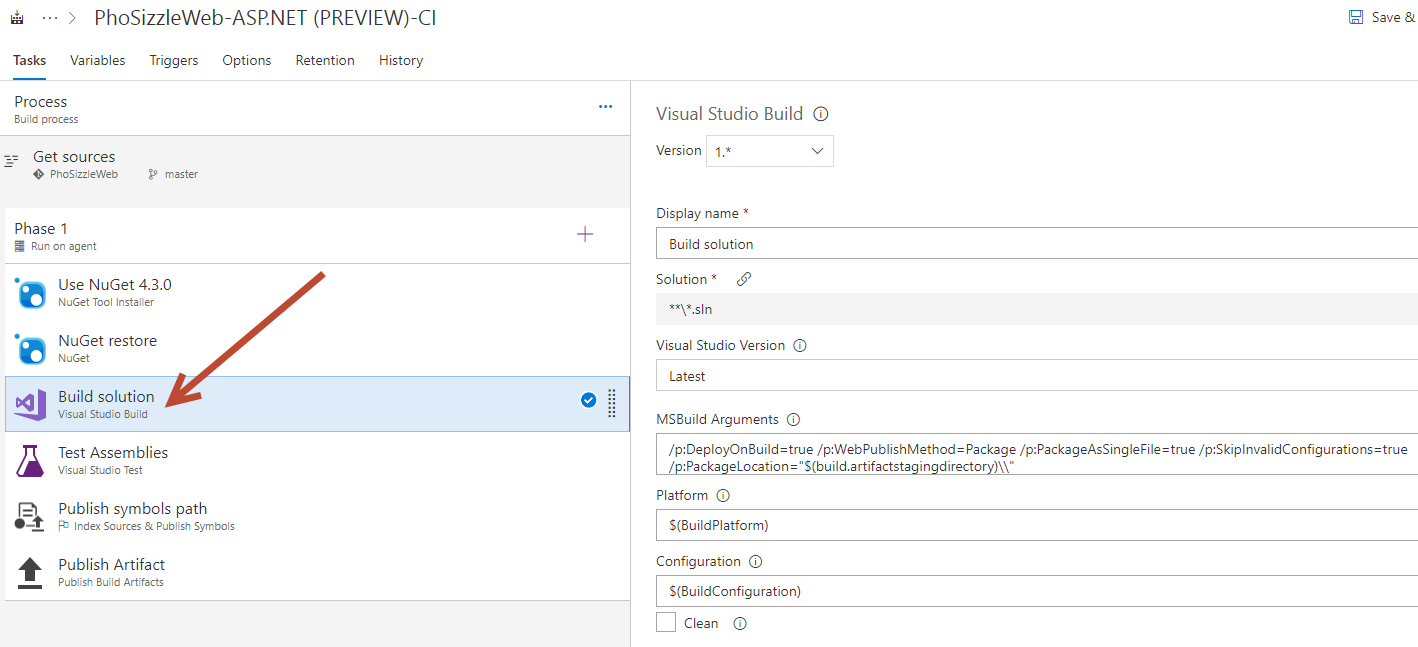
* Click on the NuGet installer task. Note what is happening, it is installing NuGet version 4.3.0 specifically. If you wanted to just get the latest NuGet version, you can check the box option but you don’t need to here.



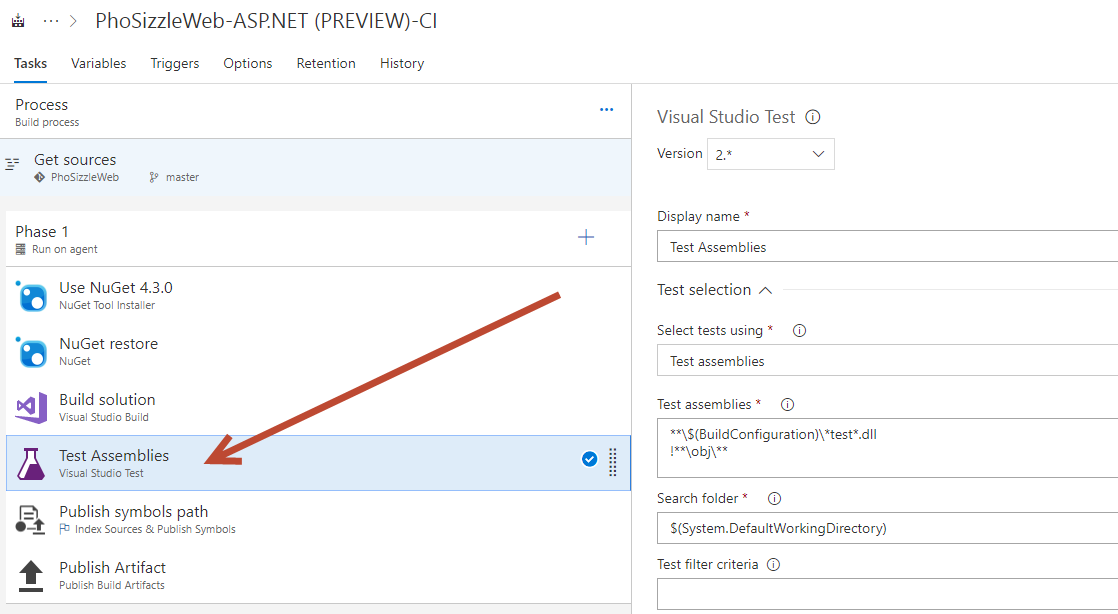
* Click on the next task for NuGet Restore. This step will restore all your NuGet packages for your solution.



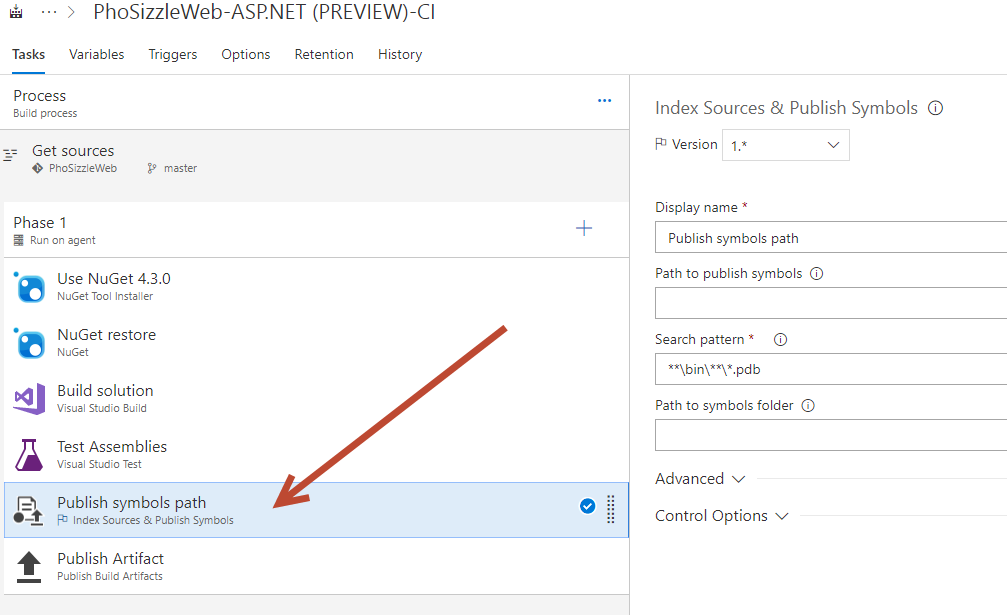
* Look at the Build solution. This is the step that will actually build your solution using Visual Studio 2017 (assuming you chose the Hosted VS 2017). You can choose a specific version of VS if needed and add on MSBuild Aruguments.



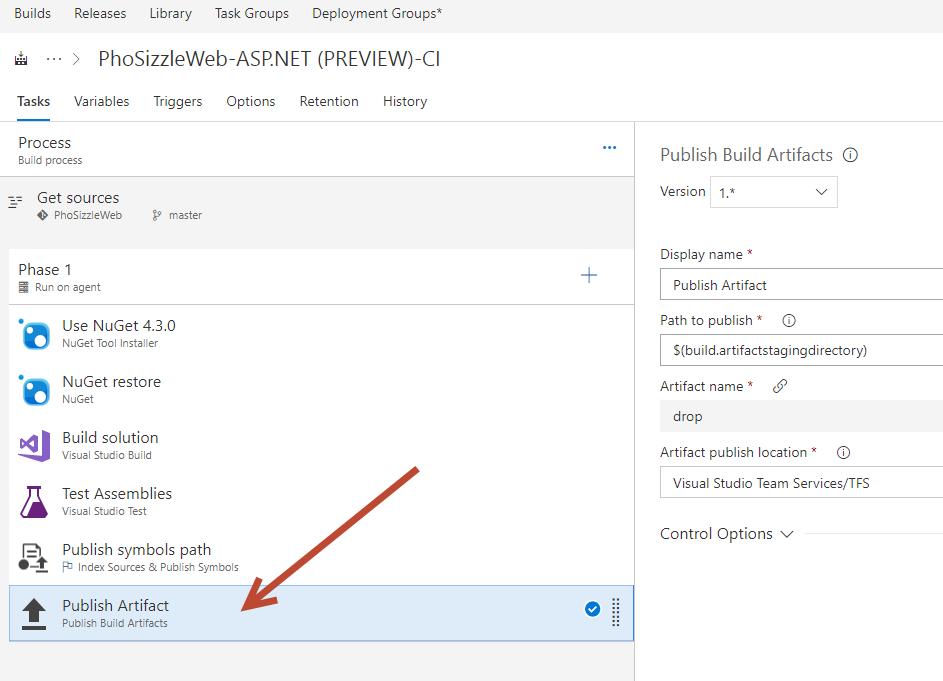
* Look at the Test Assemblies step. This will grab all assemblies with the name “test” in it by default, or you can add additional assemblies if needed to customize.

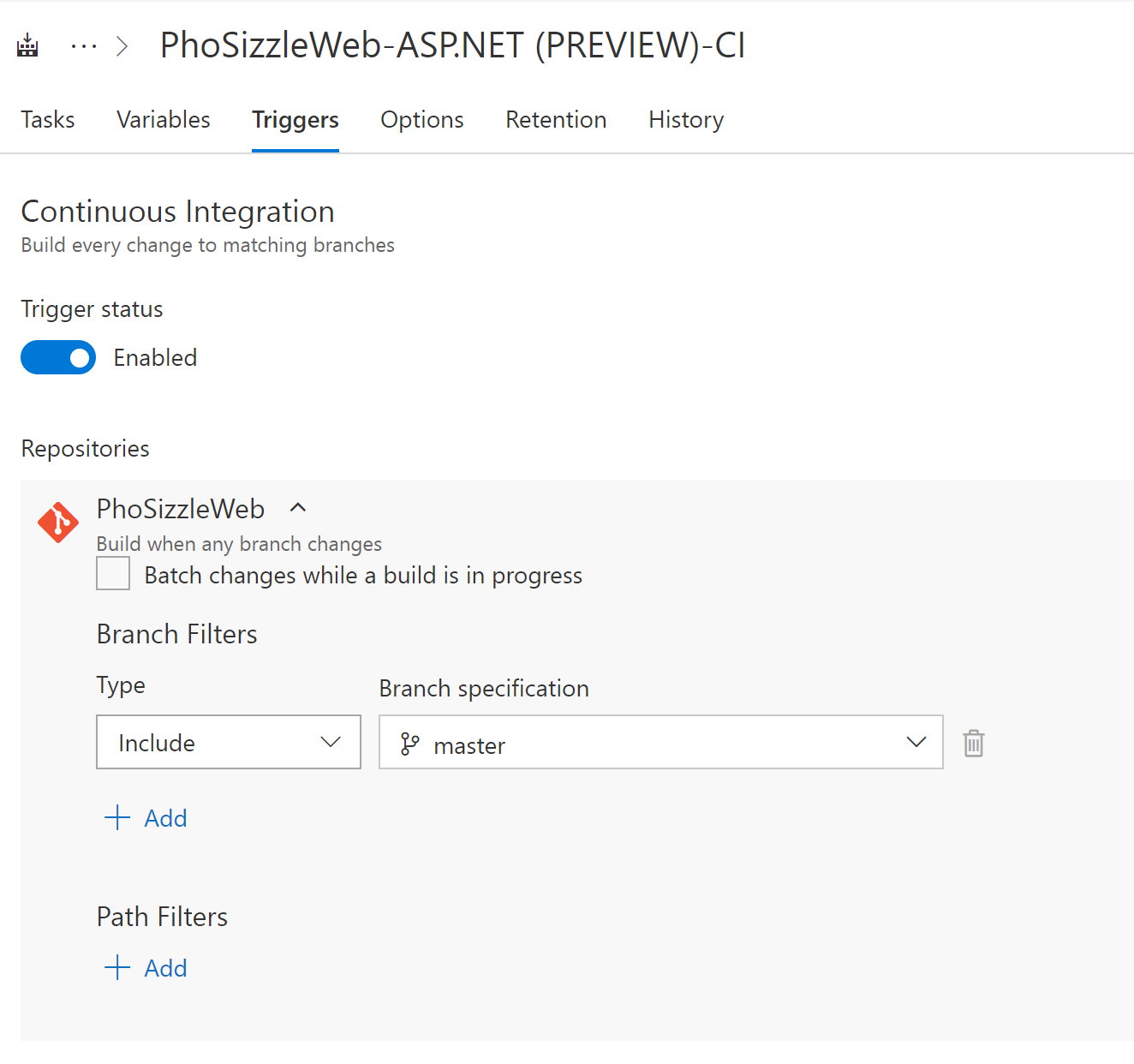


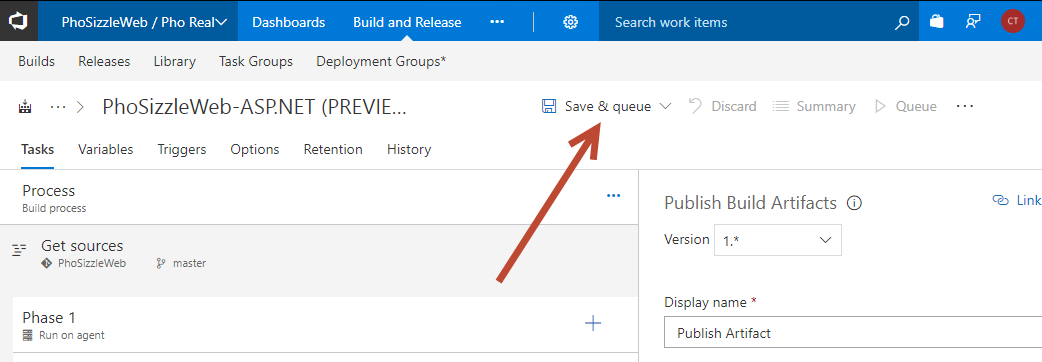
* Look at the Publish symbols step, this will discover PDF files to publish for you. You can use the .pdb symbol files to debug an app on a machine other than the one you used to build the app.



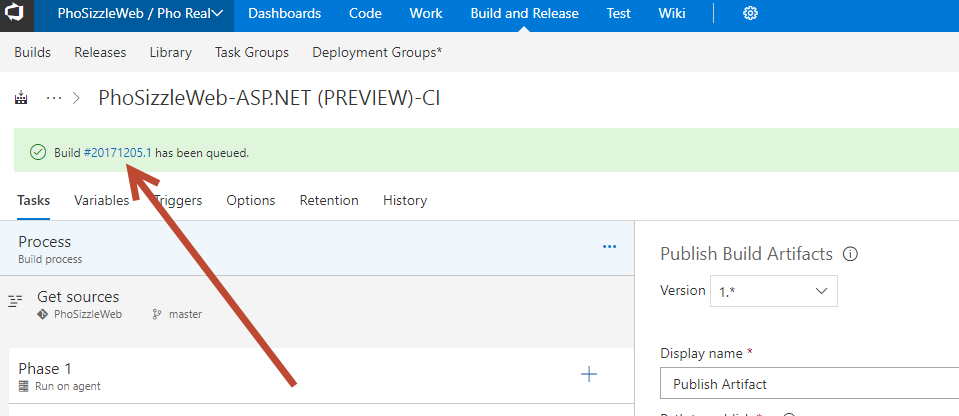
* Look at the Publish Artifact. This will publish build artifacts to Visual Studio Team Services/TFS or a file share.



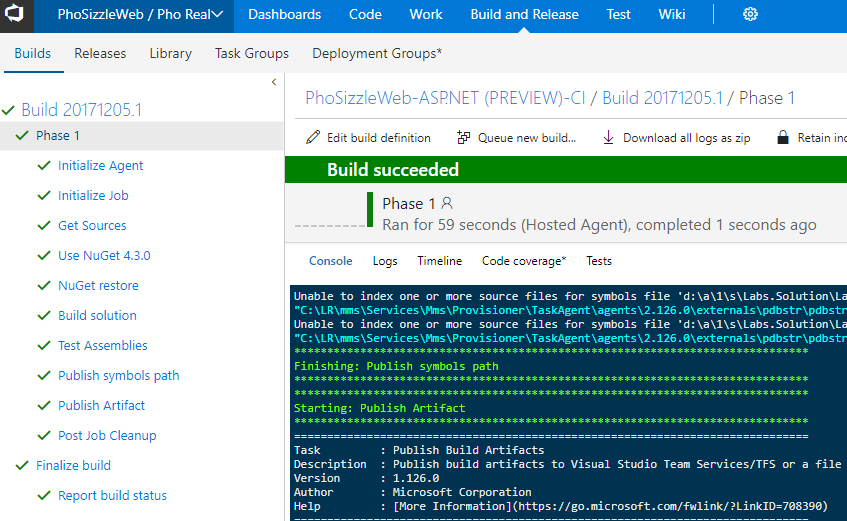
* Let’s add a trigger so any check-in to the branch will trigger a Build.
* Click on the Triggers tab. Check the tab to Enabled and now any check-in to the master branch will trigger a Build.
* 
* Hit Save and Queue



* You will see a popup where you can add a comment in like “First build”, then hit Save and Queue.
* You will see a note about the Build being queued. Click the blue text to see the Build.



* Pick the DevOpsOrg Team project, DevOpsOrg Repository and the master branch. In the next exercises we will enhance our build definition.



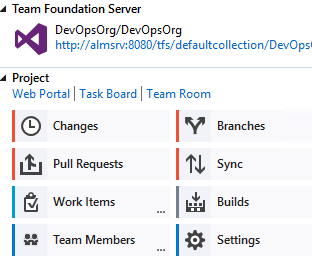
# Exercise 2 – Restoring solution packages

If you’re starting a new project, you will want to make sure you add a .gitignore file at the very beginning so that you don’t spend time cleaning your solution afterwards. The .gitignore file tells git which files (or patterns) it should ignore. It's usually used to avoid committing transient files from your working directory that aren't useful to other collaborators, such as compilation products, temporary files IDEs create, local setups, etc. It can also be used for secure or local passwords. In this exercise we will focus on an example where we want to know where to edit or add a .gitignore later in the lifetime of a solution.

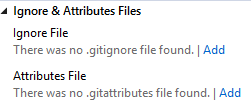
* Go to the solution on Visual Studio
* Make sure you are target the Feature Branch and not the master branch
* Right click on the solution and select **Restore Nuget Packages**.

Let’s add a .gitignore file.

* From the team explorer click on the Home icon, then the settings button:



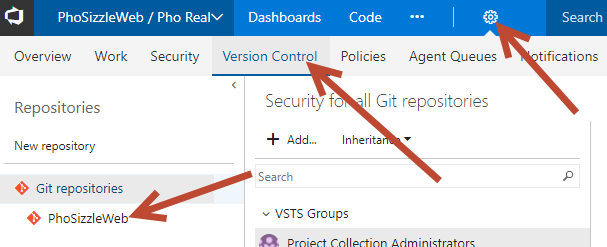
* Under Git, click Repository settings, and you can see how to edit both a .gitignore file and .gitattributes file. If they did not exist, you would see Add as in the screenshot below:



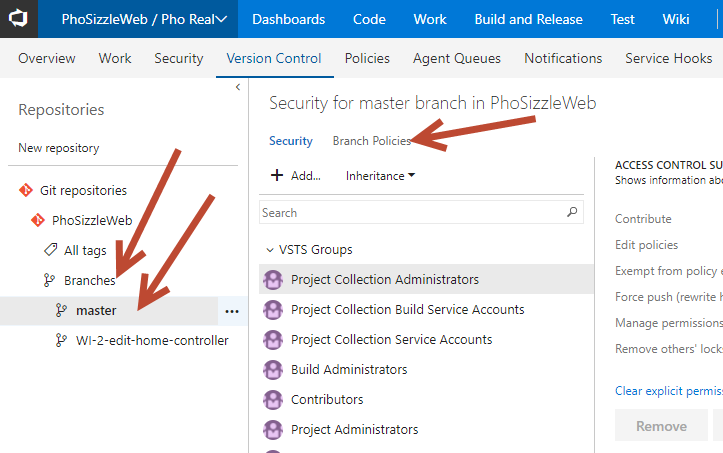
# Exercise 3 – Protecting the master branch

In the previous exercise we relied on the user triggering a pull request in order to merge changes to the master branch. This should become a habit, the merge should fail of the code hasn’t been reviewed and the build hasn’t succeeded.

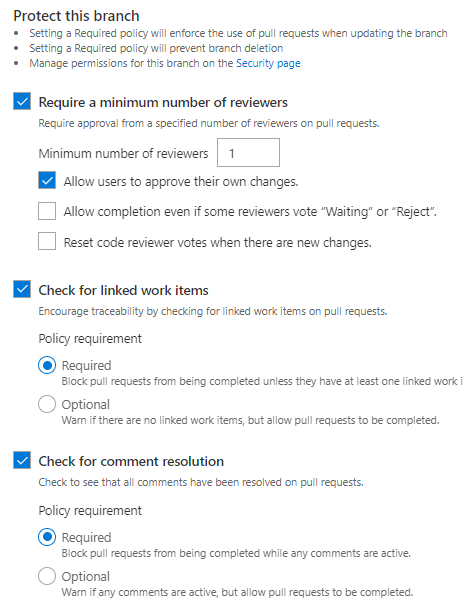
* Go the setting page for the current project. Then to the version control tab. Last, click on the name of the repository, in this case it is PhoSizzleWeb.



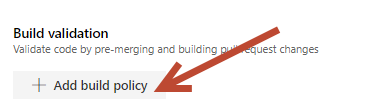
* Click the arrow to the left of Branches to expand the choices. Select the master branch, then click branch policies.



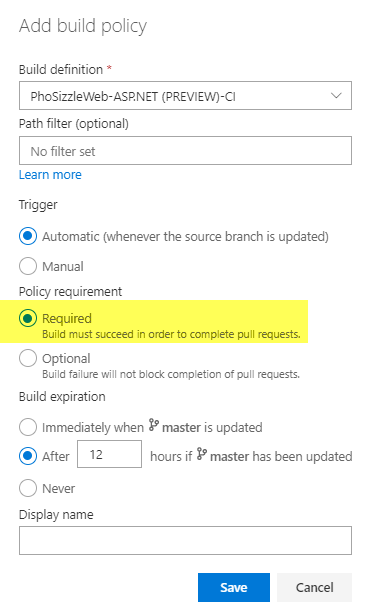
* Select the following settings:



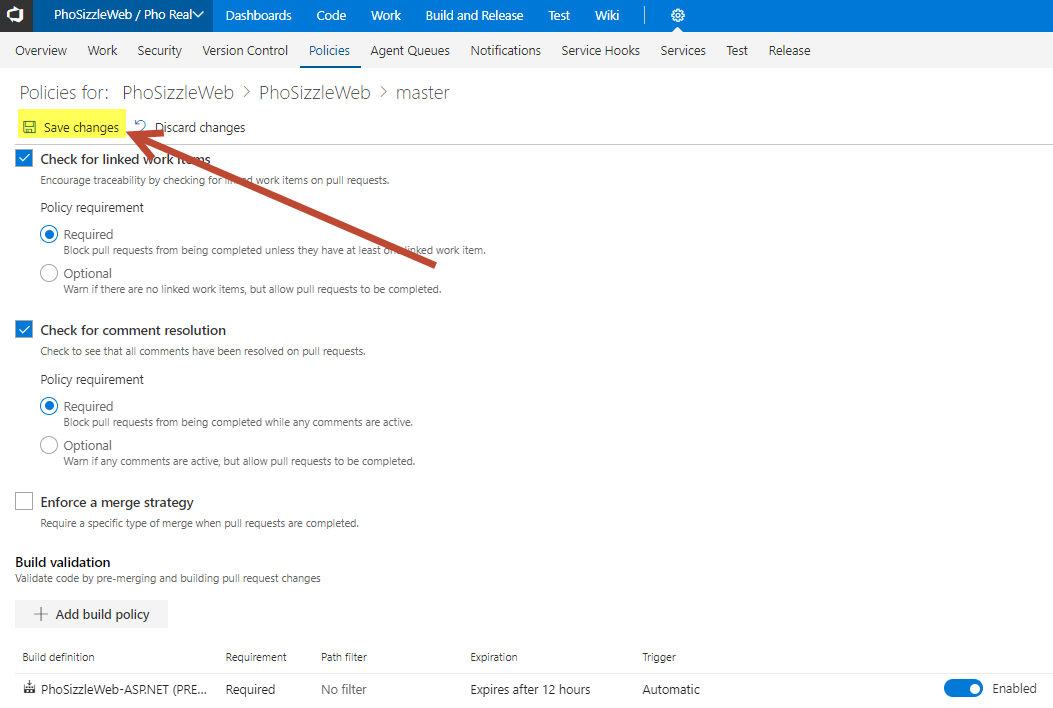
* Then, click on the setting for Build Policy.



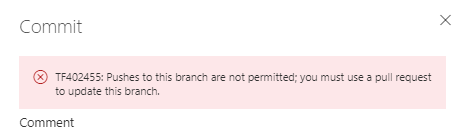
* Select the following in the modal popup, then hit Save.



* Click Save.
* Click save at the top of the page to save everything.

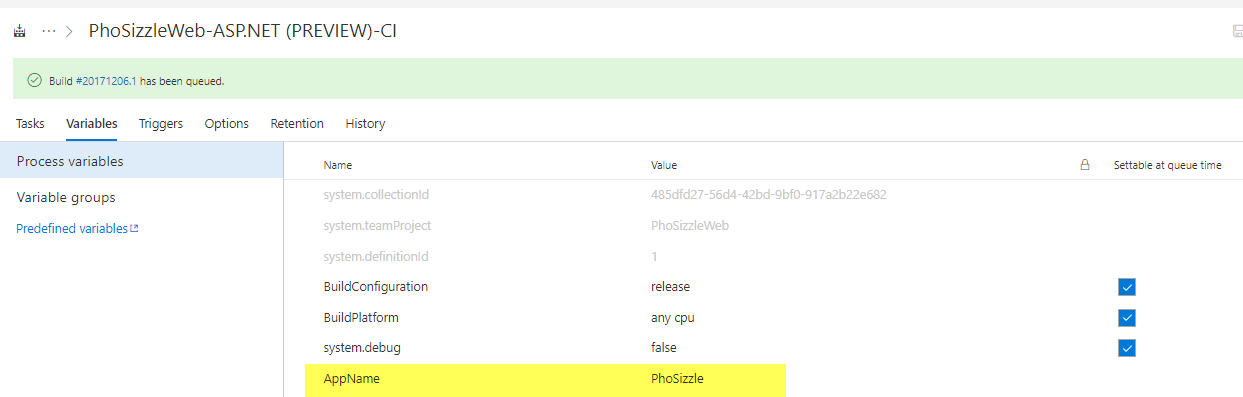


* Now your master branch is protected! You can customize these settings for each branch and project.
* To test this out, try going to your Master branch in the Code tab. Try to directly update the code in any file or add a new file using VSTS.
* You should see a warning like this when you try to commit the change:

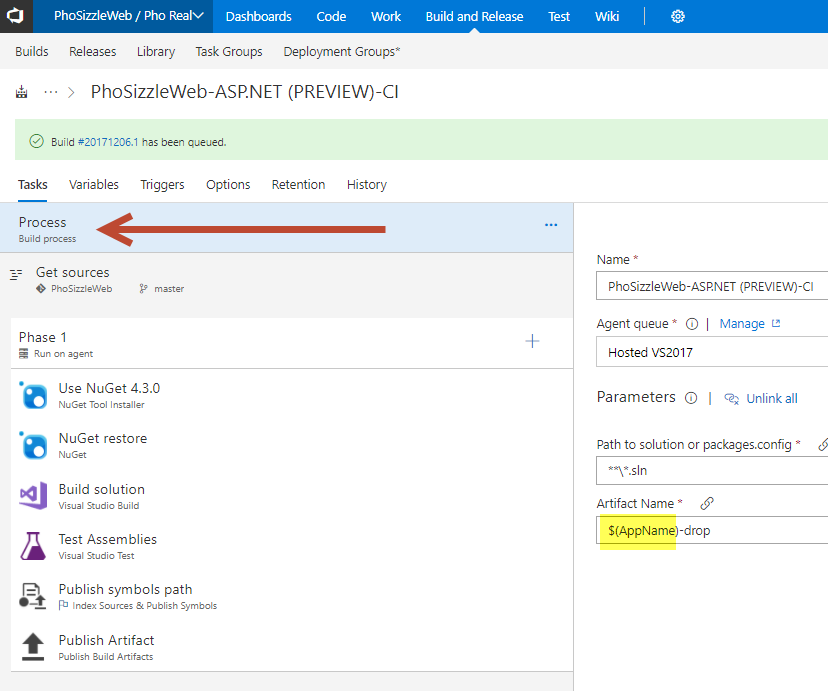


# Exercise 4 – Using variables in the build definition

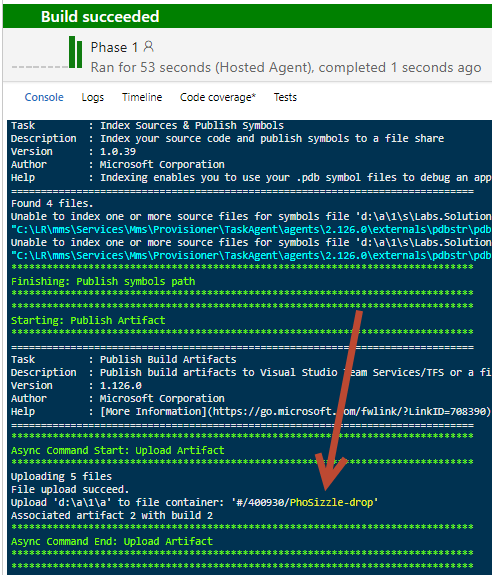
* Go back to VSTS and the Build tab.
* Then select the PhoSizzleWeb-ASP.NET (PREVIEW)-CI build definition, and click edit
* Click on the variables tab. Click Add variable, name it AppName and give it the value PhoSizzle:



* Check allow at Queue Time.
* Modify the “Copy and Publish build artifacts to use the newly added variable:



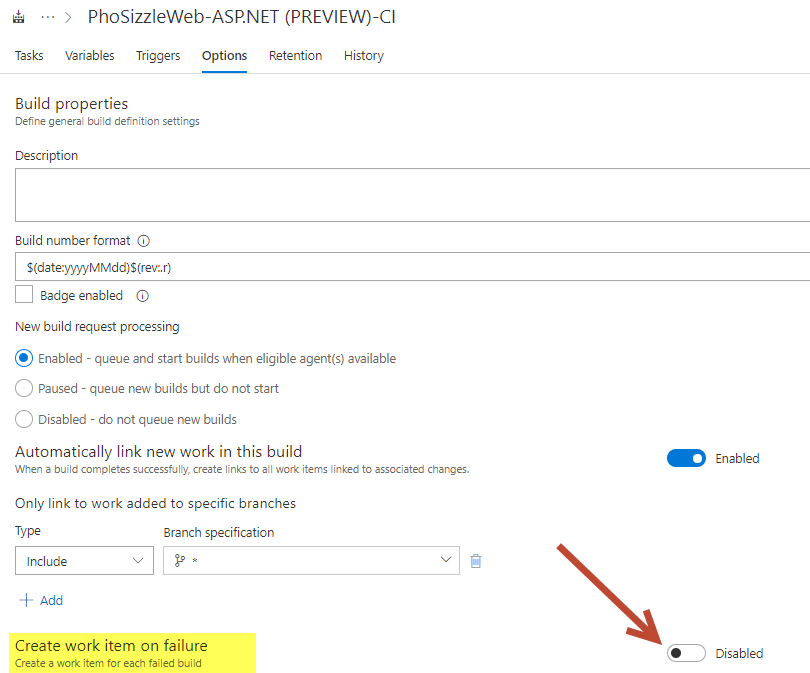
* Save the changes and queue a new build to make sure that your changes were successful.
* When the build is done, find where the build is uploaded. You should see the name is utilized now from your Variable.



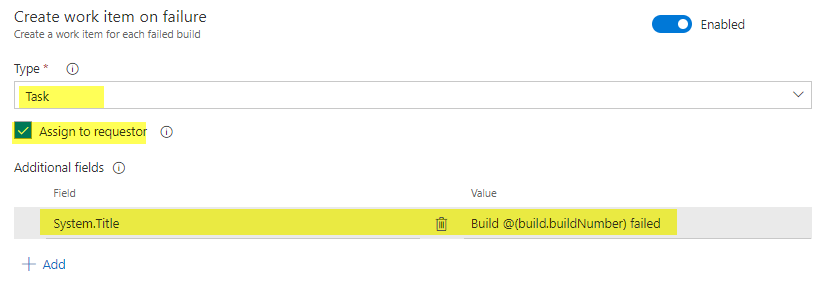
* Check out the list of all the built-in Build Variables you can utilize: <https://docs.microsoft.com/en-us/vsts/build-release/concepts/definitions/build/variables?tabs=batch>

Exercise 5 – Configuring more options:

* Edit the build configuration once more and go to the Options tab.
* Click to Enable the “Create Work Item on failure” options:



* Add the following to your enabled field.



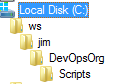
* You can also edit the Description and Build number format if desired. Under General change the build format number to: $(BuildDefinitionName)\_$(Year:yyyy).$(Month).$(DayOfMonth)$(Rev:.r)

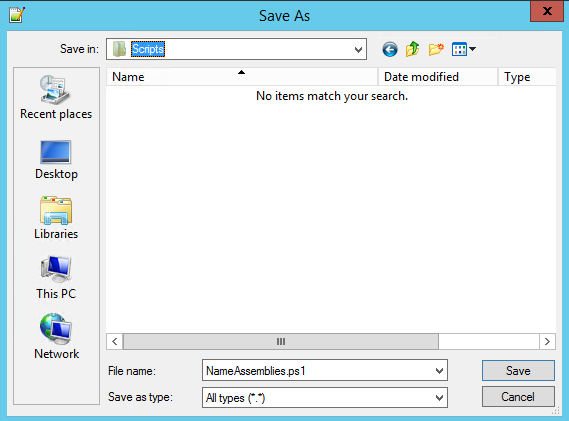


* Save the changes and queue a new build to make sure that your changes are successful. (You can potentially test your changes by introducing code that would break the build).

Exercise 6 – Executing scripts as part of the build

* VSTS build allows you to run any kind of script.
* Copy the following script into a file and save it under the current user local git repository’s folder using the following name: **NameAssemblies.ps1**
* This PS script will change file version is changed to map the build number.





##-----------------------------------------------------------------------

## <copyright file="ApplyVersionToAssemblies.ps1">(c) Microsoft Corporation. This source is subject to the Microsoft Permissive License. See http://www.microsoft.com/resources/sharedsource/licensingbasics/sharedsourcelicenses.mspx. All other rights reserved.</copyright>

##-----------------------------------------------------------------------

# Look for a 0.0.0.0 pattern in the build number.

# If found use it to version the assemblies.

#

# For example, if the 'Build number format' build process parameter

# $(BuildDefinitionName)\_$(Year:yyyy).$(Month).$(DayOfMonth)$(Rev:.r)

# then your build numbers come out like this:

# "Build HelloWorld\_2013.07.19.1"

# This script would then apply version 2013.07.19.1 to your assemblies.

# Enable -Verbose option

[CmdletBinding()]

# Regular expression pattern to find the version in the build number

# and then apply it to the assemblies

$VersionRegex = "\d+\.\d+\.\d+\.\d+"

# If this script is not running on a build server, remind user to

# set environment variables so that this script can be debugged

if(-not ($Env:BUILD\_SOURCESDIRECTORY -and $Env:BUILD\_BUILDNUMBER))

{

Write-Error "You must set the following environment variables"

Write-Error "to test this script interactively."

Write-Host '$Env:BUILD\_SOURCESDIRECTORY - For example, enter something like:'

Write-Host '$Env:BUILD\_SOURCESDIRECTORY = "C:\code\FabrikamTFVC\HelloWorld"'

Write-Host '$Env:BUILD\_BUILDNUMBER - For example, enter something like:'

Write-Host '$Env:BUILD\_BUILDNUMBER = "Build HelloWorld\_0000.00.00.0"'

exit 1

}

# Make sure path to source code directory is available

if (-not $Env:BUILD\_SOURCESDIRECTORY)

{

Write-Error ("BUILD\_SOURCESDIRECTORY environment variable is missing.")

exit 1

}

elseif (-not (Test-Path $Env:BUILD\_SOURCESDIRECTORY))

{

Write-Error "BUILD\_SOURCESDIRECTORY does not exist: $Env:BUILD\_SOURCESDIRECTORY"

exit 1

}

Write-Verbose "BUILD\_SOURCESDIRECTORY: $Env:BUILD\_SOURCESDIRECTORY"

# Make sure there is a build number

if (-not $Env:BUILD\_BUILDNUMBER)

{

Write-Error ("BUILD\_BUILDNUMBER environment variable is missing.")

exit 1

}

Write-Verbose "BUILD\_BUILDNUMBER: $Env:BUILD\_BUILDNUMBER"

# Get and validate the version data

$VersionData = [regex]::matches($Env:BUILD\_BUILDNUMBER,$VersionRegex)

switch($VersionData.Count)

{

0

{

Write-Error "Could not find version number data in BUILD\_BUILDNUMBER."

exit 1

}

1 {}

default

{

Write-Warning "Found more than instance of version data in BUILD\_BUILDNUMBER."

Write-Warning "Will assume first instance is version."

}

}

$NewVersion = $VersionData[0]

Write-Verbose "Version: $NewVersion"

# Apply the version to the assembly property files

$files = gci $Env:BUILD\_SOURCESDIRECTORY -recurse -include "\*Properties\*","My Project" |

?{ $\_.PSIsContainer } |

foreach { gci -Path $\_.FullName -Recurse -include AssemblyInfo.\* }

if($files)

{

Write-Verbose "Will apply $NewVersion to $($files.count) files."

foreach ($file in $files) {

$filecontent = Get-Content($file)

attrib $file -r

$filecontent -replace $VersionRegex, $NewVersion | Out-File $file

Write-Verbose "$file.FullName - version applied"

}

}

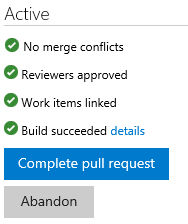
else

{

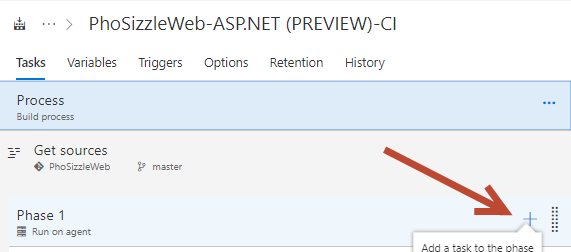
Write-Warning "Found no files."

}

* Go to Visual Studio, Stage, Commit and push the PowerShell file to the Feature branch repository.
* Submit a pull request to merge it with the master branch.
* Approve the Pull request.
* Complete the Pull Request to merge it into the Master branch.

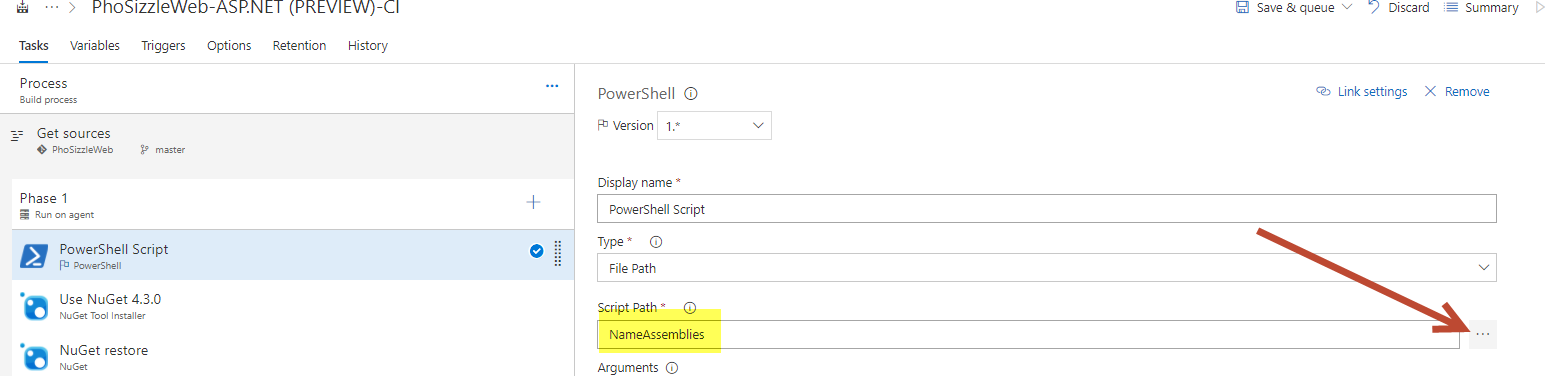


* Go back to the build definition and add a task

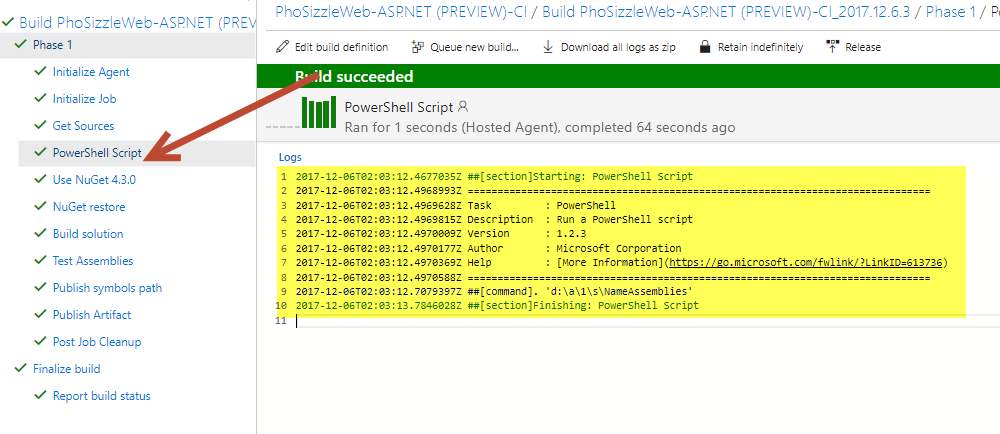


* The task should be PowerShell task. Drag and drop the tasks so that the PowerShell one is the first task in the build definition, then point it to the newly added PowerShell script.





* Save and Queue a build to make sure that your changes were successful. You can check the Build for a success and that the PowerShell step ran with no issue.



# Exercise 7 – Extending the build tasks from the VSTS Marketplace

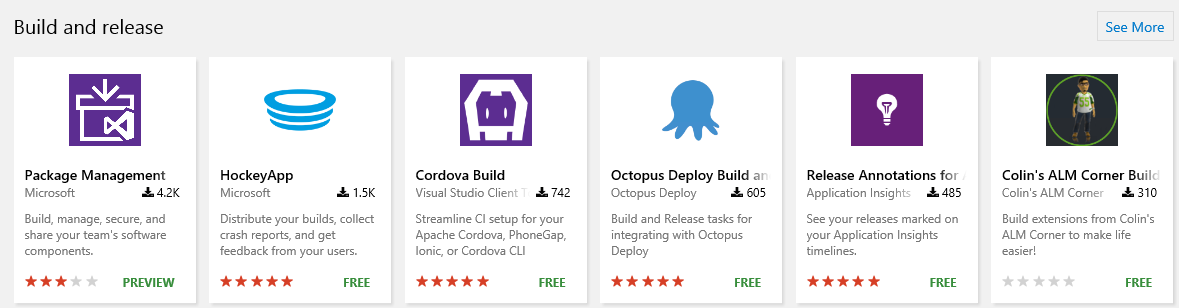
The list of available tasks can be extended by adding additional tasks from the Visual Studio Team Services marketplace.

* Let’s go to the Marketplace for VSTS:

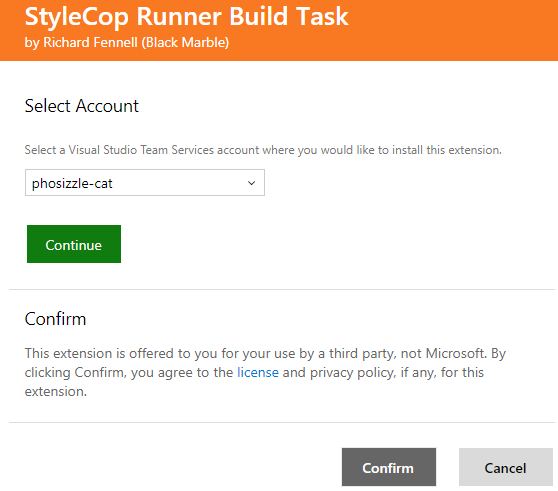
<https://marketplace.visualstudio.com/vsts>

* You can scroll down to Build and Release or go to this URL to see specific ones to Build and Release:

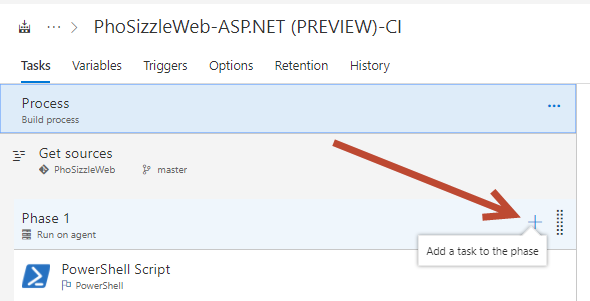
<https://marketplace.visualstudio.com/search?target=VSTS&category=Build%20and%20release&sortBy=Downloads>



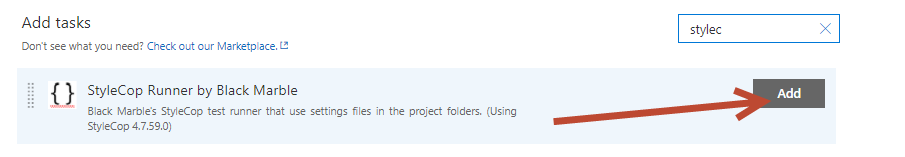
* Look up the StyleCop Runner build task. You can also directly go to the extension page: <https://marketplace.visualstudio.com/items?itemName=richardfennellBM.BM-VSTS-StyleCop-Task>
* Click Install (you will need to sign in using your Microsoft account associated with your VSTS, or it will get it automatically)
* Choose your correct project to install it, hit Continue then Confirm



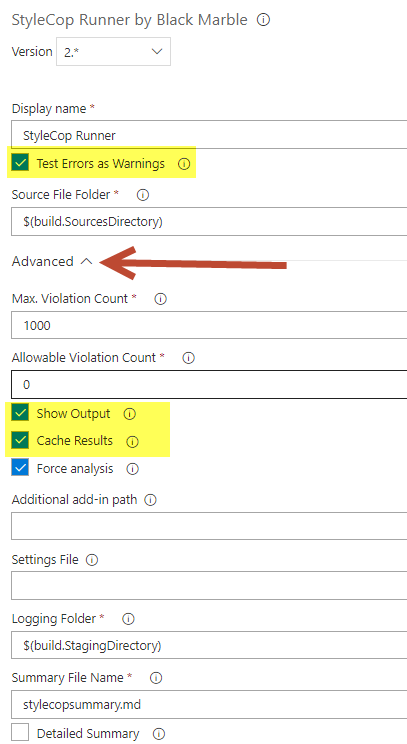
* Modify the Build definition to utilize this new extension, hit Add Task.



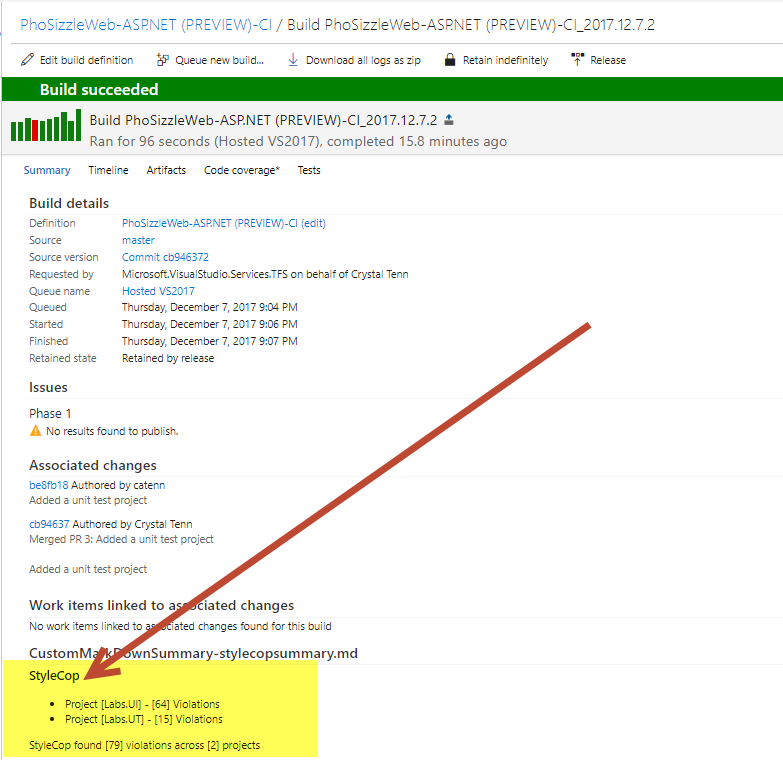
* Add your new StyleCop Task



* Add it as the last task in the Build
* Use the following settings (don’t forget to hit Advanced to get the extra settings):

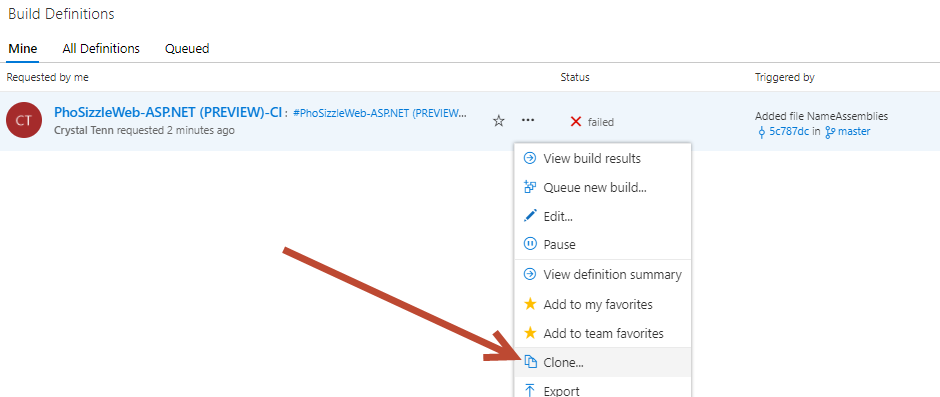


* Save the changes and queue a new build to make sure your changes were successful. You will see on your Build Summary a StyleCop Summary.



# Exercise 8 – Cloning build definitions

* Go to the build hub.
* Use the “…” menu next to the build definition you created to launch the actions menu. Choose Clone.



* Name the build: **PhoSizzleWeb-Dev-CI**
* Hit Save and Queue. This is a simple way to make builds for other environments that are almost identical. You can change small things as needed like triggers / options / variables etc.