Workshop DevOps V2

Prerequisites:

Git download link:

<https://git-scm.com/download/>

Visual Studio Code:­

<https://code.visualstudio.com/Download>

Visual Studio account (create and login):

<https://www.visualstudio.com/>

Dotnet core SDK 2.1.403:

<https://www.microsoft.com/net/download/core#/sdk>

[Optional] Docker community edition:

<https://www.docker.com/community-edition>

Info:

Azure portal link: <https://portal.azure.com>

Steps:

1. Start Visual Studio Code and press: “*CTRL + `”* to bring up the console prompt
2. Type: *dotnet new mvc* -> create a new MVC application based on a predefined template
3. Test the application with: *dotnet restore* // *dotnet build* // *dotnet run* -> you will see it run on <http://localhost:5000>
4. Add the following line in the *Program.cs* file: .UseUrls(“http://\*:5000”) – this will make the application listen on port 5000 (all IPs). Should look something like this:

public static IWebHost BuildWebHost(string[] args) =>

WebHost.CreateDefaultBuilder(args)

.UseStartup<Startup>()

.UseUrls("http://\*:5000")

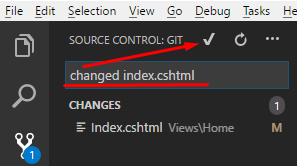
.Build();

1. Create new Visual Studio Team Services project using the UI from the *<username>.visualstudio.com* portal. (This requires a Visual Studio account)
2. Type in VS Code: *git init* -> initialize the local git repository
3. Type in VS Code: *git add .* -> add the current directory (“.”) as changes to the local repository
4. Type in VS Code: g*it commit -m "Commit message"* or, from Visual Studio Source Control tab, type the message and commit. -> commit the changes to the local repository
5. Type in VS Code: *git remote add origin https://<****username****>.visualstudio.com/\_git/<****project****>* -> add the remote location
6. Type in VS Code: *git push -u origin –all* -> push the changes to the remote location (use git credential manager to enter username and password)
7. Check VSTS (Visual Studio Team Services) to see that the master Branch has been created and we have files there. (Menu: *Code -> Files*)

Create a new branch and merge:

***Note: normally you in a development process you should not work directly with the master branch, it should only be updated for example when a feature is completed.***

1. Type in VS Code: *git branch -m branch1* -> create and move to branch ***<branch1>***
2. Change application code -> add or change text in, for example, *Views/Home/index.cshtml* file
3. Commit the changes from UI as explained in point 7 (See image)



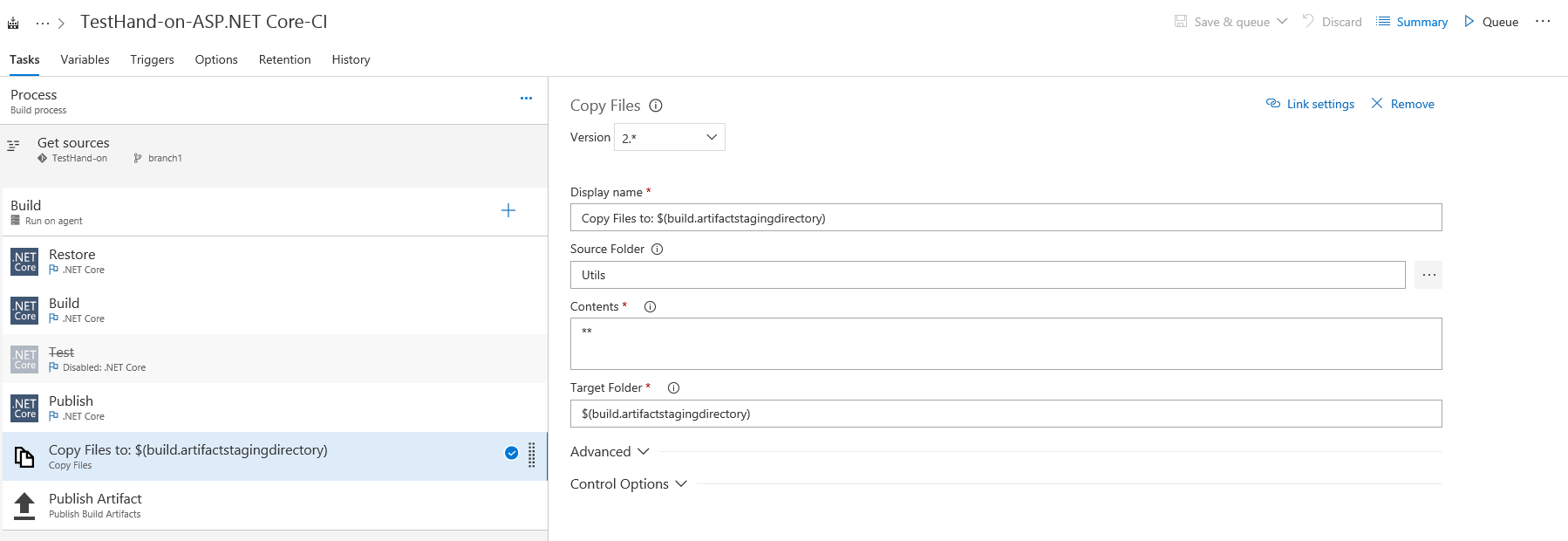
1. Type in VS Code: *git push origin branch1* -> push the new branch to the remote one
2. **From VSTS portal:** 
   1. Go to builds
   2. Start with build definition -> ASP.NET core build definition
   3. Set trigger for continuous integration -> triggers tab
   4. Create a pull request to get the new branch code into master from the portal -> Code > Pull Requests > New Pull Request
   5. Approve and complete the merge
3. This triggered a build. Check builds you should see the running build. You will be able to see cool logs with the steps taken to complete the build.
4. Go to Azure Portal
5. Create a Dev Test Lab (new -> devtest labs -> fill in the information and create)
6. Create a machine in the created Dev Tests Lab (tested on windows 2016 Datacenter)
   1. Create a secret we will be using (optional)
   2. Fill in the required information
   3. [required] Please make sure you use a static, public IP. Otherwise we will need to take into consideration a load balancer.
   4. Get the ARM template when you create the machine (last step)
7. Go back to Visual Studio Code.
8. Create a new folder and file (in the folder) in the project using VS Code and paste the template there
9. Copy all the required scripts and files (dockerfile) in the same folder. Also, check them out and in the dotnet.bat and dockerfile please rename the DLL file with the name of the application you created (same name as the folder you created it in). ***Example: from VSTSTest.dll to <my new application.dll>***
10. Push the code to git (directly to master):
    1. *git add .*
    2. g*it commit -m "Commit message"*
    3. *git push -u origin –all*
11. **Go to the VSTS portal:**
    1. Create a task in the build definition: **Copy files**
    2. Configure it and add the target folder as:

**$(build.artifactstagingdirectory)**

And contents: **\*\* - all the files in the folder**

* 1. Run a build so we can have a sample of the files in the artefacts directory

The build should look like this in the end (all tasks except copy files should be default ones):



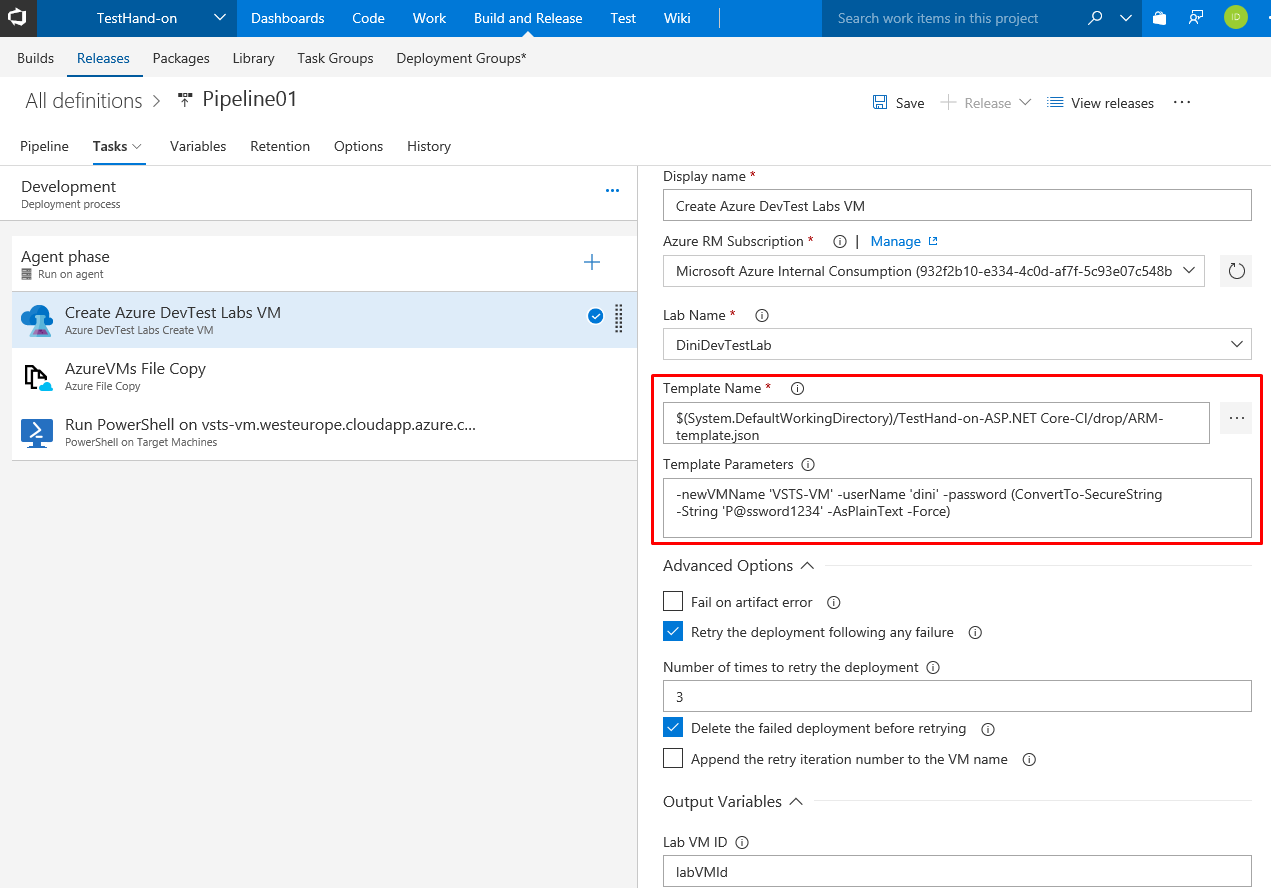
* 1. **Create a new empty release pipeline**
  2. In the UI: Add artifact to start from - the previously created build – latest
  3. Set the continuous integration trigger
  4. Got to tasks and create: **Create Azure DevTest Labs VM**

1. Set-up the configuration in the task and replace the template parameters with the proper names. See (highlighted is the name of the ARM template file created earlier):

“**$(System.DefaultWorkingDirectory)/$(Build.DefinitionName)/drop/<ARM-TemplateFile>.json**”

or “**$(System.ArtifactsDirectory)/$(Build.DefinitionName)/drop/<ARM-TemplateFile>.json**”

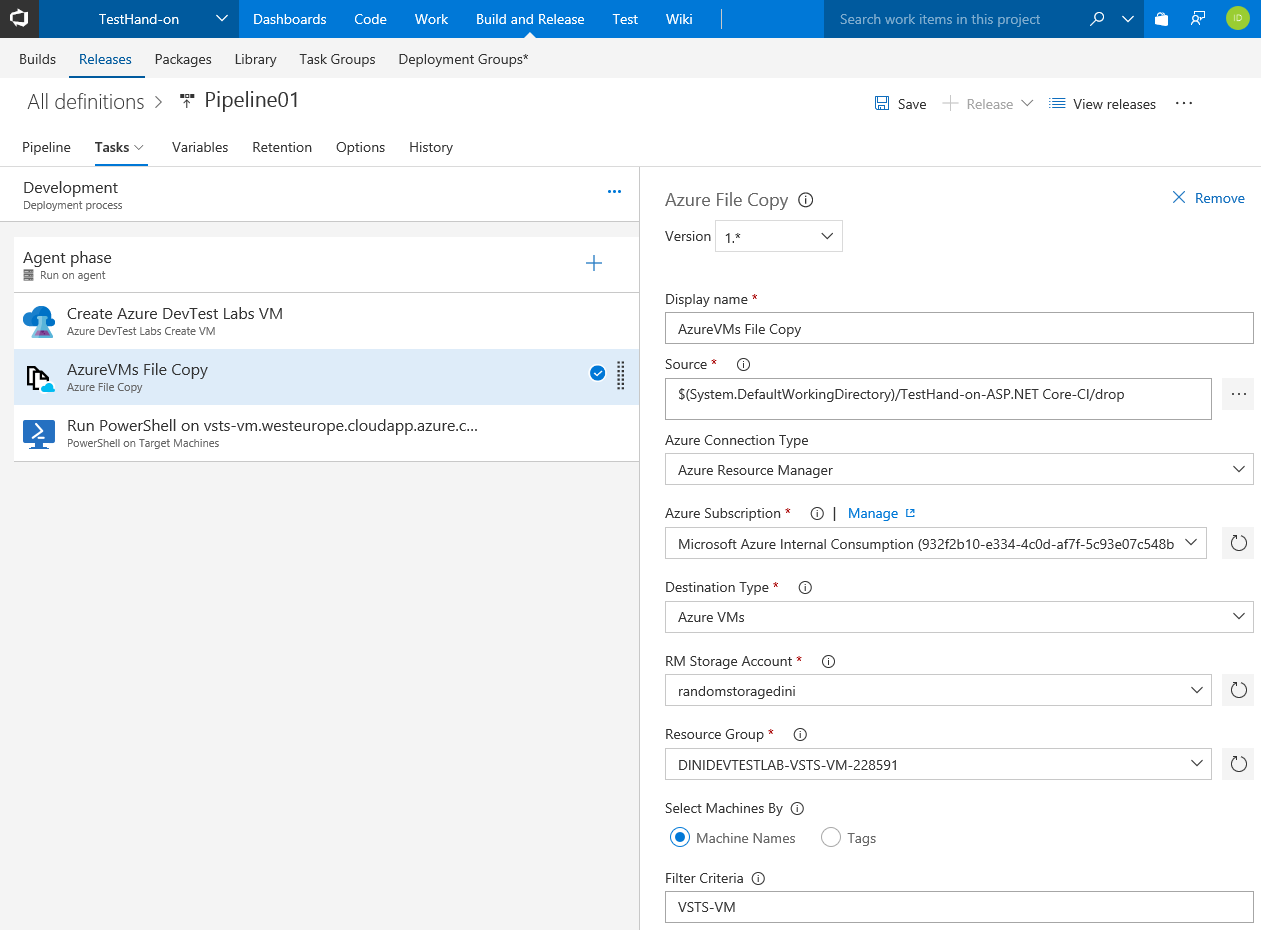
The task should look like this (high attention area in red):

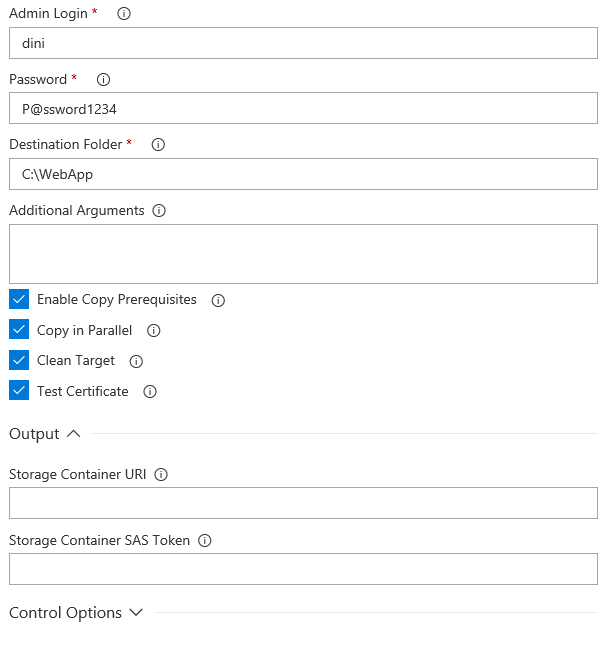


1. Create a new release and deploy. The machine should get created and you should be able to see it in the Azure Portal.
2. Create task **Azure File Copy** and set it up (for VMs) using the information you entered when you created the dev test lab Virtual Machine. You should be able to find the information in the portal or in the previous task. Default folder should be: C:\WebApp as it is used in the scripts.

***NOTE: As explained in the lab the path and many other variables can be kept directly in VSTS and used as variables so they will not be hardcoded. They are available as environment variables on the agent machines also.***

Should look like this after configuration:





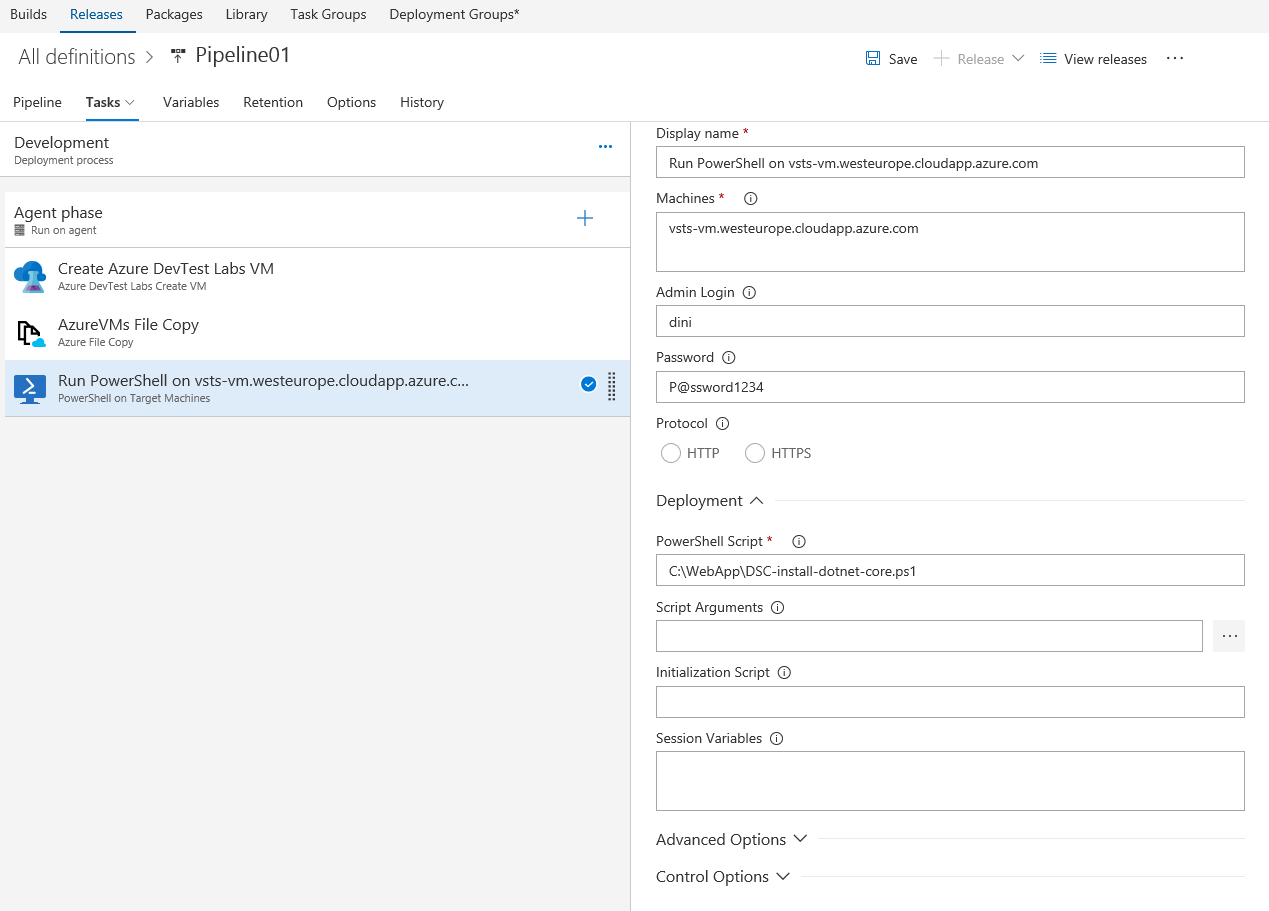
1. Run a release again. (now, we will have the application files on the machine that we created earlier).
2. RDP to VM created in the Dev Test Labs and check the files are there (using the connect button in the azure portal).
3. Information regarding the scripts (the files should already be on the virtual machine created).
   1. Fist one (***dotnet-install-2.ps1***) is public and only copy-pasted and changed the dotnet core version to 1.1 from this github repository:

V2:

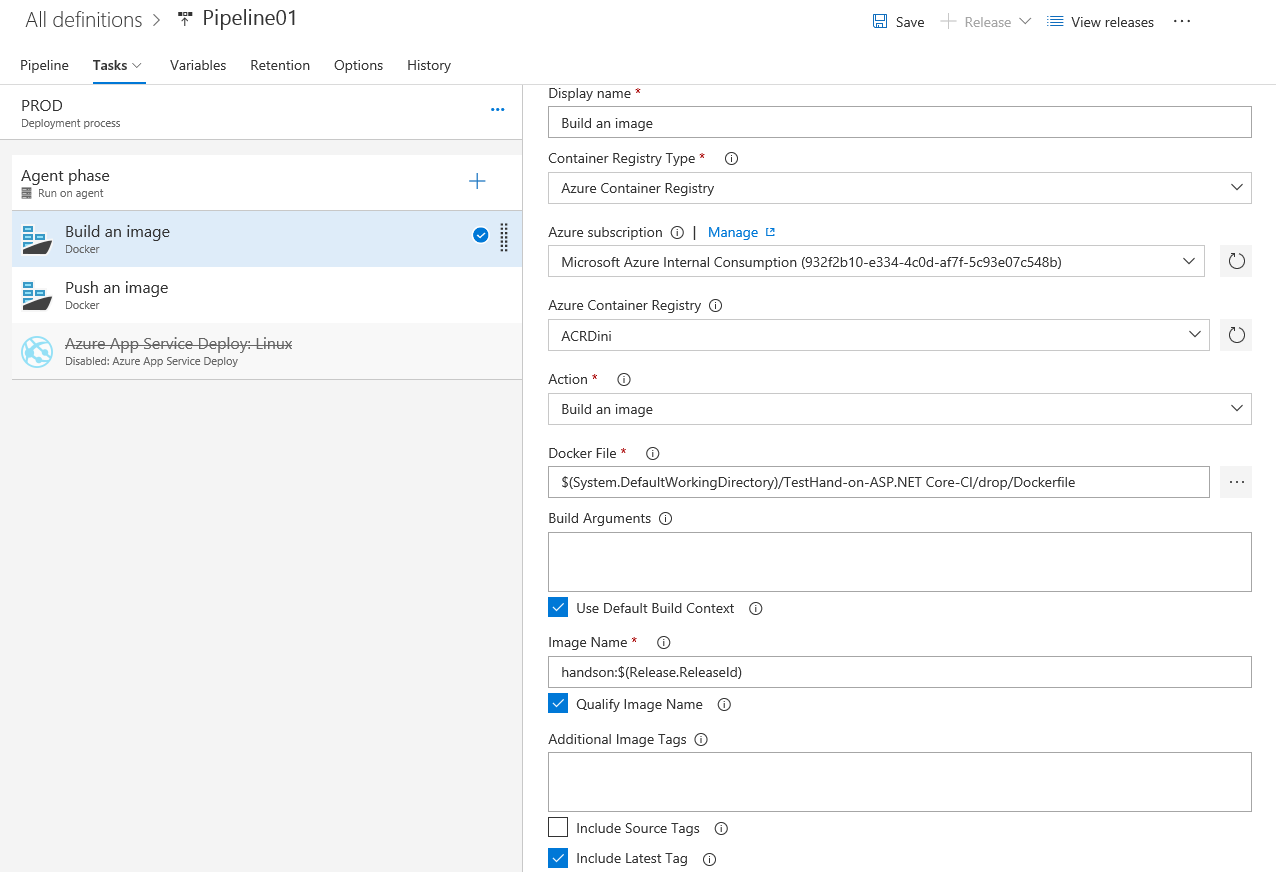
<https://github.com/dotnet/docs/blob/master/docs/core/tools/dotnet-install-script.md>

* 1. The second (***runApp.ps1***) created by me to unzip the files and run the application.
  2. The third (***dotnet.bat***) is used to run in a background mode so VSTS can receive the exit callback and continue with the release pipeline.
  3. [Optional] Change the Firewall rules to make sure we have external connectivity. If applicable also the load balancer rules and NSG rules.
  4. **DSC-install-dotnet-core.ps1 –** DSC (Desired State Configuration) script to orchestrate the installation of dotnet core SDK 2.1.3 and application start. (It’s set to use the ***current*** channel, so you might need to specify the version if the latest is newer than 2.1.3)

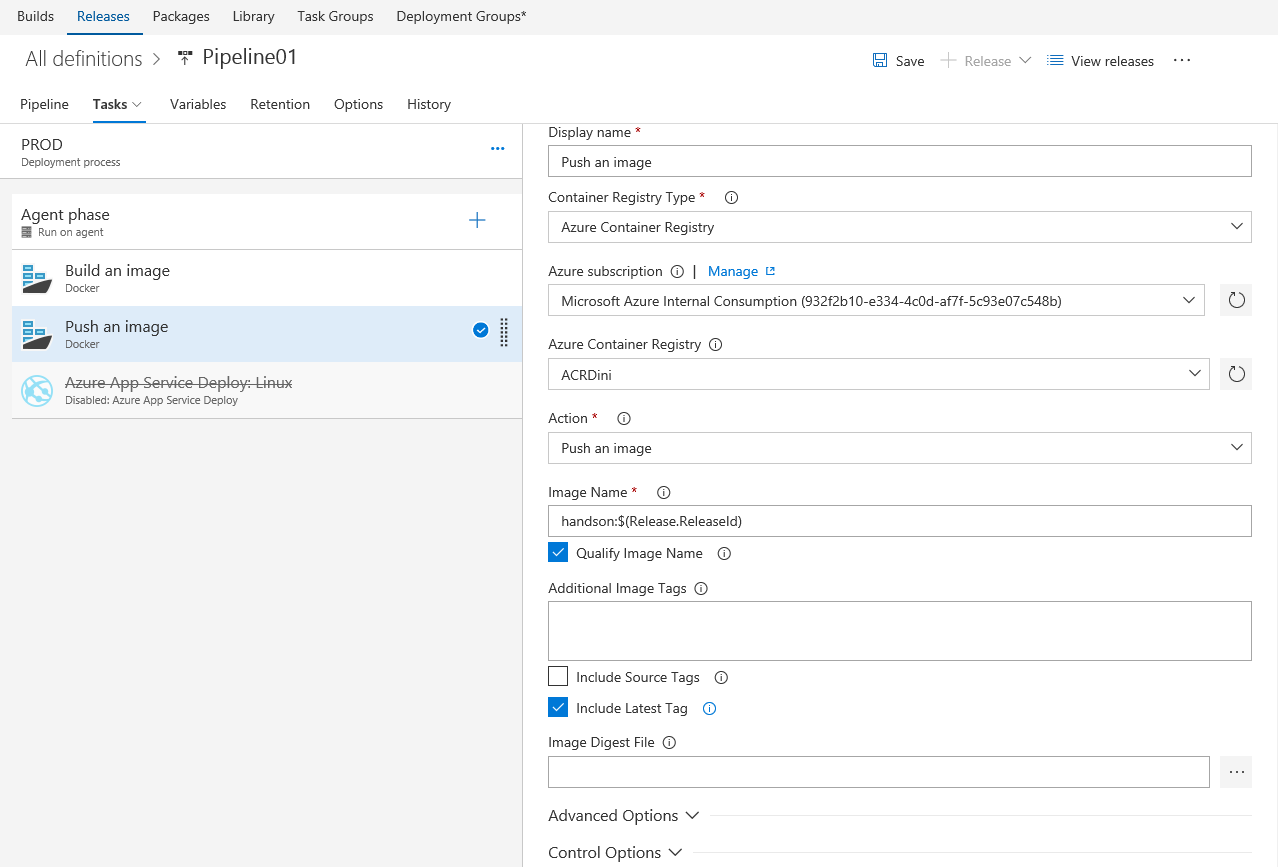
1. Go to VSTS portal
2. Create ***PowerShell on target machines*** task and configure (FQDN for name). Should look like this:



1. Go to Azure portal and **create a new Web App on Windows**
2. Back in the VSTS portal
3. **Create the Q&A Environment and use the template: Azure App Service Deployment**
4. Configure the environment (Select subscription and web app created earlier)
5. [Optional] Disable the first environment to make the process faster.
6. Create a new release to deploy in the web app also
7. Go to Azure portal – Check if the application hosted in the web app on windows is available.
8. **Azure Portal: Create a webapp on linux**
9. **Go to VSTS Portal and create the PROD environment**
10. For the tasks in this environment please use the linux VSTS agent because it will contain the needed linux dependencies (like docker).
11. The dockerfile should already be in the pipeline
12. Add a Docker task: Build an image



1. Add a Docker task: Push an image



1. Go to the Azure Portal
2. Set PORT application setting with the value 5000 -> in the App Settings area
3. Go to the docker section and select the image you want to be used on the Web App on linux
4. Start a new build process and check the last environment and maybe the whole pipeline.