Devops

**Azure Pipeline**

Trigger > Build > QA > Production

**Stage** – series of jobs

**Job**- series of steps

**Step** – Task (prepackaged script to perform action) or script (powershell/bash script)

Agent is a vm 🡪 where job runs to build your code.

In pipeline, you specify underlying image which is required to build your code.

**In visual studio**

Create a new project > asp.net core (name) > .net core > create

New repo> azurewebapp

Connect this repo with visual studio

In visual studio > git> existing remote > place url > push code

Code changes

Pages > index.html >

# ASP.NET Core (.NET Framework)

# Build and test ASP.NET Core projects targeting the full .NET Framework.

# Add steps that publish symbols, save build artifacts, and more:

# https://docs.microsoft.com/azure/devops/pipelines/languages/dotnet-core

trigger:

- master

pool:

  vmImage: 'windows-latest'

variables:

  solution: '\*\*/\*.sln'

  buildPlatform: 'Any CPU'

  buildConfiguration: 'Release'

jobs:

- job: BuildJob

  steps:

  - bash: echo "Build Pipeline for AzureWebApps"

  - task: NuGetToolInstaller@1

  - task: NuGetCommand@2

    inputs:

      restoreSolution: '$(solution)'

  - task: VSBuild@1

    inputs:

      solution: '$(solution)'

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

      platform: '$(buildPlatform)'

      configuration: '$(buildConfiguration)'

**Self-hosted agent (software)**

Install git.exe – require to checkout code

Install nuget.exe – require restoring the packages for .net solution

Install visual studio and .net core for build .net solution

Azure pipelines agent

**Agent pools**

Azure Pipeline – used for Microsoft agents

Default- anyone for selfhosted agent

1. Spin vm
2. Turn off internet security
3. Download git and visual studio code

Default pool > new agent

1. Goto vm and sign onto vm and do the setting up of agent on the machine itself.
2. Follow instructions on vm
3. Enter server url ( https://dev.azure.com/achrooorg/) {server/organization}
4. Goto security > Generate PAT (personal access token) > Add PAT into self hosted agent command prompt

**Modify pipeline to use self hosted agent**

# ASP.NET Core (.NET Framework)

# Build and test ASP.NET Core projects targeting the full .NET Framework.

# Add steps that publish symbols, save build artifacts, and more:

# https://docs.microsoft.com/azure/devops/pipelines/languages/dotnet-core

trigger:

- master

pool:

name: ‘My Pool for pipelines’ {agent pool name}

  vmImage: 'winvm' {agent name}

variables:

  solution: '\*\*/\*.sln'

  buildPlatform: 'Any CPU'

  buildConfiguration: 'Release'

jobs:

- job: BuildJob

  steps:

  - bash: echo "Build Pipeline for AzureWebApps"

  - task: NuGetToolInstaller@1

  - task: NuGetCommand@2

    inputs:

      restoreSolution: '$(solution)'

  - task: VSBuild@1

    inputs:

      solution: '$(solution)'

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

      platform: '$(buildPlatform)'

      configuration: '$(buildConfiguration)'

Install Jenkins on Window Server

Download java11

Download Jenkins

Go to local security policy> Local Policies > User Right Assignment > Log on as a service

Graphical user interface, text, application

Description automatically generated

Double click

Add user in group

Install Jenkins > use creds while setup

netsh advfirewall firewall add rule name="TCP Port 8080" dir=in action=allow protocol=TCP localport=8080

Allow port 8080 in nsg

**Installing required components of Jenkins**

Git download

Visual studio – azure development and asp.net core

Nuget– nuget.org - C:\Program Files\nuget (put exe file here or any other location)

**Configuration of Jenkins**

Manage Jenkins > manage plugins > available plugins > msbuild (msbuild is installed with visual studio) but plugin is required to interact with server

Global tool configuration >

goto git > add location of git.exe

**C:\Program Files\Git\bin\git**

goto msbuild > add location of ms build path found in visual studio path

**C:\Program Files\Microsoft Visual Studio\2022\Community\MSBuild\Current\Bin**

goto configure system- note Home directory path

C:\ProgramData\Jenkins\.jenkins {artifacts will be stored in this path} {programdata is hidden folder}

Reverse proxy error can be removed by changing Jenkins url

C:\Program Files\nuget\nuget.exe restore “C:\ProgramData\Jenkins\.jenkins\workspace\app-project\GitAzureWebApp.sln”

**Whitesource bolt**

Code scan and vulnerabilities

Goto organization settings > extension > white source bolt (in extension section, after installation, u need to create free account for 30 days white source)

Add white source bolt task in pipeline at end

U can see white source bolt report

1. # ASP.NET Core (.NET Framework)
2. # Build and test ASP.NET Core projects targeting the full .NET Framework.
3. # Add steps that publish symbols, save build artifacts, and more:
4. # https://docs.microsoft.com/azure/devops/pipelines/languages/dotnet-core
6. trigger:
7. - master
9. pool:
10. vmImage: 'windows-latest'
12. variables:
13. solution: '\*\*/\*.sln'
14. buildPlatform: 'Any CPU'
15. buildConfiguration: 'Release'
17. steps:
18. - task: NuGetToolInstaller@1
20. - task: NuGetCommand@2
21. inputs:
22. restoreSolution: '$(solution)'
24. - task: VSBuild@1
25. inputs:
26. solution: '$(solution)'
27. msbuildArgs: '/p:DeployOnBuild=true /p:WebPublishMethod=Package /p:PackageAsSingleFile=true /p:SkipInvalidConfigurations=true /p:DesktopBuildPackageLocation="$(build.artifactStagingDirectory)\WebApp.zip" /p:DeployIisAppPath="Default Web Site"'
28. platform: '$(buildPlatform)'
29. configuration: '$(buildConfiguration)'
31. - task: WhiteSource Bolt@20
32. # ASP.NET Core (.NET Framework)
33. # Build and test ASP.NET Core projects targeting the full .NET Framework.
34. # Add steps that publish symbols, save build artifacts, and more:
35. # https://docs.microsoft.com/azure/devops/pipelines/languages/dotnet-core
37. trigger:
38. - master
40. pool:
41. vmImage: 'windows-latest'
43. variables:
44. solution: '\*\*/\*.sln'
45. buildPlatform: 'Any CPU'
46. buildConfiguration: 'Release'
48. steps:
49. - task: NuGetToolInstaller@1
51. - task: NuGetCommand@2
52. inputs:
53. restoreSolution: '$(solution)'
55. - task: VSBuild@1
56. inputs:
57. solution: '$(solution)'
58. msbuildArgs: '/p:DeployOnBuild=true /p:WebPublishMethod=Package /p:PackageAsSingleFile=true /p:SkipInvalidConfigurations=true /p:DesktopBuildPackageLocation="$(build.artifactStagingDirectory)\WebApp.zip" /p:DeployIisAppPath="Default Web Site"'
59. platform: '$(buildPlatform)'
60. configuration: '$(buildConfiguration)'
62. - task: DotNetCoreCLI@2
63. inputs:
64. command: test
65. projects: '\*\*/\*Test/\*.csproj'
66. arguments: '--configuration $(buildConfiguration)'
67. trigger:
68. - master
70. pool:
71. vmImage: 'windows-latest'
73. variables:
74. solution: '\*\*/\*.sln'
75. buildPlatform: 'Any CPU'
76. buildConfiguration: 'Release'
78. steps:
79. - task: NuGetToolInstaller@1
81. - task: NuGetCommand@2
82. inputs:
83. restoreSolution: '$(solution)'
85. - task: VSBuild@1
86. inputs:
87. solution: '$(solution)'
88. msbuildArgs: '/p:DeployOnBuild=true /p:WebPublishMethod=Package /p:PackageAsSingleFile=true /p:SkipInvalidConfigurations=true /p:DesktopBuildPackageLocation="$(build.artifactStagingDirectory)\WebApp.zip" /p:DeployIisAppPath="Default Web Site"'
89. platform: '$(buildPlatform)'
90. configuration: '$(buildConfiguration)'
92. - task: DotNetCoreCLI@2
93. displayName: 'dotnet test'
94. inputs:
95. command: 'test'
96. projects: '\*\*/\*Test.csproj'
97. arguments: '/p:CollectCoverage=true /p:CoverletOutputFormat=cobertura /p:CoverletOutput=./MyCoverage/'
98. publishTestResults: true
100. - task: PublishCodeCoverageResults@1
101. displayName: 'Publish Code Coverage Results'
102. inputs:
103. codeCoverageTool: 'Cobertura'
104. summaryFileLocation: '$(Build.SourcesDirectory)/\*\*/MyCoverage/coverage.cobertura.xml'
105. failIfCoverageEmpty: true
106. # ASP.NET Core (.NET Framework)
107. # Build and test ASP.NET Core projects targeting the full .NET Framework.
108. # Add steps that publish symbols, save build artifacts, and more:
109. # https://docs.microsoft.com/azure/devops/pipelines/languages/dotnet-core
111. trigger:
112. - master
114. pool:
115. vmImage: 'windows-latest'
117. variables:
118. solution: '\*\*/\*.sln'
119. buildPlatform: 'Any CPU'
120. buildConfiguration: 'Release'
122. steps:
123. - task: NuGetToolInstaller@1
125. - task: NuGetCommand@2
126. inputs:
127. restoreSolution: '$(solution)'
129. - task: SonarCloudPrepare@1
130. inputs:
131. SonarCloud: 'sonar-connection'
132. organization: 'app-org'
133. scannerMode: 'MSBuild'
134. projectKey: 'app-project'
135. projectName: 'app-project'
136. - task: VSBuild@1
137. inputs:
138. solution: '$(solution)'
139. msbuildArgs: '/p:DeployOnBuild=true /p:WebPublishMethod=Package /p:PackageAsSingleFile=true /p:SkipInvalidConfigurations=true /p:DesktopBuildPackageLocation="$(build.artifactStagingDirectory)\WebApp.zip" /p:DeployIisAppPath="Default Web Site"'
140. platform: '$(buildPlatform)'
141. configuration: '$(buildConfiguration)'
143. - task: SonarCloudAnalyze@1
144. - task: SonarCloudPublish@1
145. inputs:
146. pollingTimeoutSec: '300'

start management service and allow remote mgmt.

.net core hosting bundle, web deploy on vm

Open port 8172

Assign dns name

Table

Description automatically generated

Custom script extension only once. Powershell desired state configuration it also run once. But it can be combined with Azure Automation and can be used again. If IIS is uninstalled, then azure automation will again reinstall.

**Powershell DSC**

We have two files

1. WebRole.ps1 – zip this file (mandatory), it contains configuration name (mentioned after Configuration keyword, we will be using this)
2. InstallWebconfig.psd1 - data configuration file

Upload both files (zip file and psd1 file) in container of storage account

Then go to extension search for powershell (u will find powershell desire state configuration)

Graphical user interface, application

Description automatically generated

**Azure Automation**

Create automation account

Graphical user interface, text, application

Description automatically generated

Upload your powershell script

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, application

Description automatically generated

Compile script in automation

Graphical user interface, text, application, email

Description automatically generated

Associate configuration with node

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

**Powershell DSC for other machines**

On-permise and other cloud environment

For windows machine, window mgmt. framework 5 shud be installed

**For linux machines**

Ensure powershell dsc for linux is installed on machine.

Use register.py command on machine to register itself with azure automation acct

Generate dsc metaconfigurations

Apply metaconfiguration settings

1. Create a file name NewConfig.ps1 file and add the following contents

1. configuration NewConfig
2. {
3. Node AllNodes
4. {
5. WindowsFeature IIS
6. {
7. Ensure = 'Present'
8. Name = 'Web-Server'
9. IncludeAllSubFeature = $true
10. }
11. }
12. }

2. Create a file called config.ps1 and add the following contents. Run the file on the target machine. Ensure to change aspects such as the Automation Account URL and the key

Once you executed the config.ps1 file, it will generate a folder

Then execute the following command

Set-DscLocalConfigurationManager -Path ./DscMetaConfigs

1. # The DSC configuration that will generate metaconfigurations
2. [DscLocalConfigurationManager()]
3. Configuration DscMetaConfigs
4. {
5. param
6. (
7. [Parameter(Mandatory=$True)]
8. [String]$RegistrationUrl,
10. [Parameter(Mandatory=$True)]
11. [String]$RegistrationKey,
13. [Parameter(Mandatory=$True)]
14. [String[]]$ComputerName,
16. [Int]$RefreshFrequencyMins = 30,
18. [Int]$ConfigurationModeFrequencyMins = 15,
20. [String]$ConfigurationMode = 'ApplyAndMonitor',
22. [String]$NodeConfigurationName,
24. [Boolean]$RebootNodeIfNeeded= $False,
26. [String]$ActionAfterReboot = 'ContinueConfiguration',
28. [Boolean]$AllowModuleOverwrite = $False,
30. [Boolean]$ReportOnly
31. )
33. if(!$NodeConfigurationName -or $NodeConfigurationName -eq '')
34. {
35. $ConfigurationNames = $null
36. }
37. else
38. {
39. $ConfigurationNames = @($NodeConfigurationName)
40. }
42. if($ReportOnly)
43. {
44. $RefreshMode = 'PUSH'
45. }
46. else
47. {
48. $RefreshMode = 'PULL'
49. }
51. Node $ComputerName
52. {
53. Settings
54. {
55. RefreshFrequencyMins = $RefreshFrequencyMins
56. RefreshMode = $RefreshMode
57. ConfigurationMode = $ConfigurationMode
58. AllowModuleOverwrite = $AllowModuleOverwrite
59. RebootNodeIfNeeded = $RebootNodeIfNeeded
60. ActionAfterReboot = $ActionAfterReboot
61. ConfigurationModeFrequencyMins = $ConfigurationModeFrequencyMins
62. }
64. if(!$ReportOnly)
65. {
66. ConfigurationRepositoryWeb AzureAutomationStateConfiguration
67. {
68. ServerUrl = $RegistrationUrl
69. RegistrationKey = $RegistrationKey
70. ConfigurationNames = $ConfigurationNames
71. }
73. ResourceRepositoryWeb AzureAutomationStateConfiguration
74. {
75. ServerUrl = $RegistrationUrl
76. RegistrationKey = $RegistrationKey
77. }
78. }
80. ReportServerWeb AzureAutomationStateConfiguration
81. {
82. ServerUrl = $RegistrationUrl
83. RegistrationKey = $RegistrationKey
84. }
85. }
86. }
88. # Create the metaconfigurations
89. # NOTE: DSC Node Configuration names are case sensitive in the portal.
90. # TODO: edit the below as needed for your use case
91. $Params = @{
92. RegistrationUrl = 'https://48c66e92-f2eb-41b5-b656-d80714edd870.agentsvc.ne.azure-automation.net/accounts/48c66e92-f2eb-41b5-b656-d80714edd870';
93. RegistrationKey = 'mqVTA2tgXC2yrvDIr/anS1TUQdWfSUmjVKFJc5+z9uvgNrwR/WW9UbE1bqBwZvj0bQzJrzoHd9dmRlX/vHTrHw==';
94. ComputerName = @('demovm');
95. NodeConfigurationName = 'NewConfig.AllNodes';
96. RefreshFrequencyMins = 30;
97. ConfigurationModeFrequencyMins = 15;
98. RebootNodeIfNeeded = $False;
99. AllowModuleOverwrite = $False;
100. ConfigurationMode = 'ApplyAndMonitor';
101. ActionAfterReboot = 'ContinueConfiguration';
102. ReportOnly = $False; # Set to $True to have machines only report to AA DSC but not pull from it
103. }
105. # Use PowerShell splatting to pass parameters to the DSC configuration being invoked
106. # For more info about splatting, run: Get-Help -Name about\_Splatting
107. DscMetaConfigs @Params

**ARM Template**

Version of template language being used - $schema

version of template – contentVersion (revision of template) “”

Collection of api version for resource types – api profile “”

Values that can be provided during deployment – parameters {}

Values that can be reused in template – variables {}& functions []

Resources that need to be deployed – resources []

Values that can be retrieved after deployment – output {}

**Continuous Delivery/Deployment**

Delivery -Pipeline should be able to deploy application into production-based environment

Deployment – entire process from code commit to production is automated