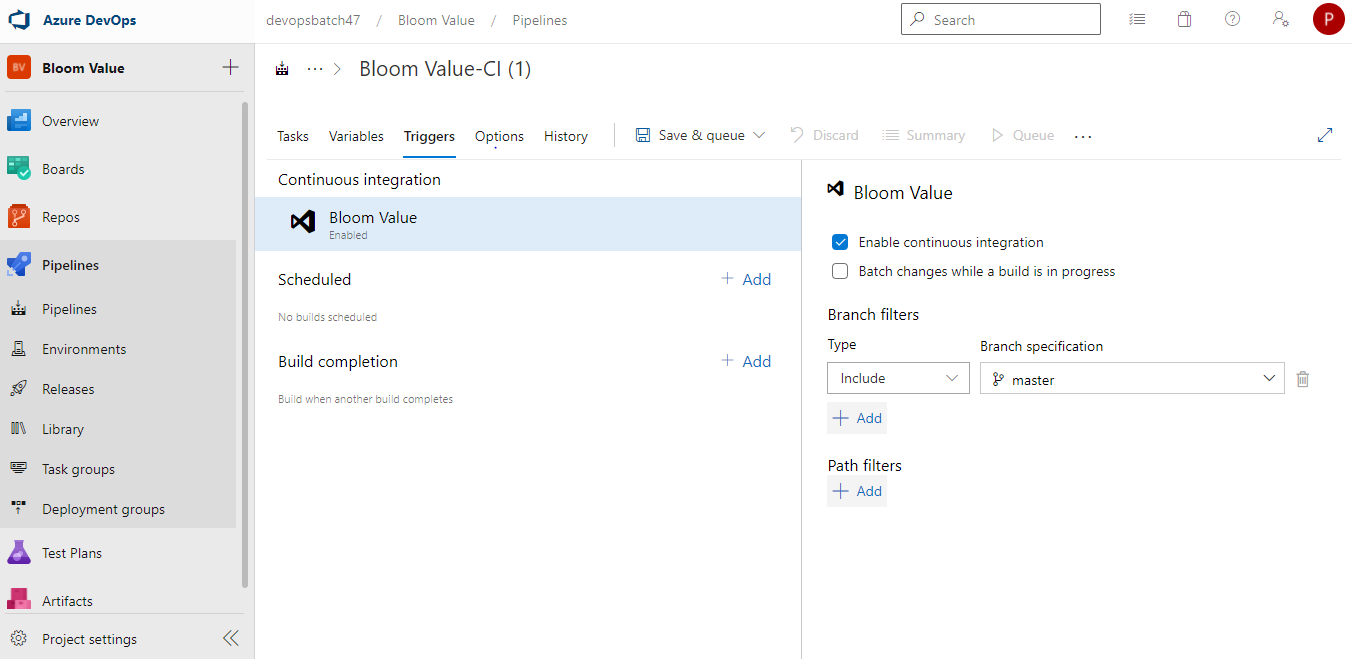
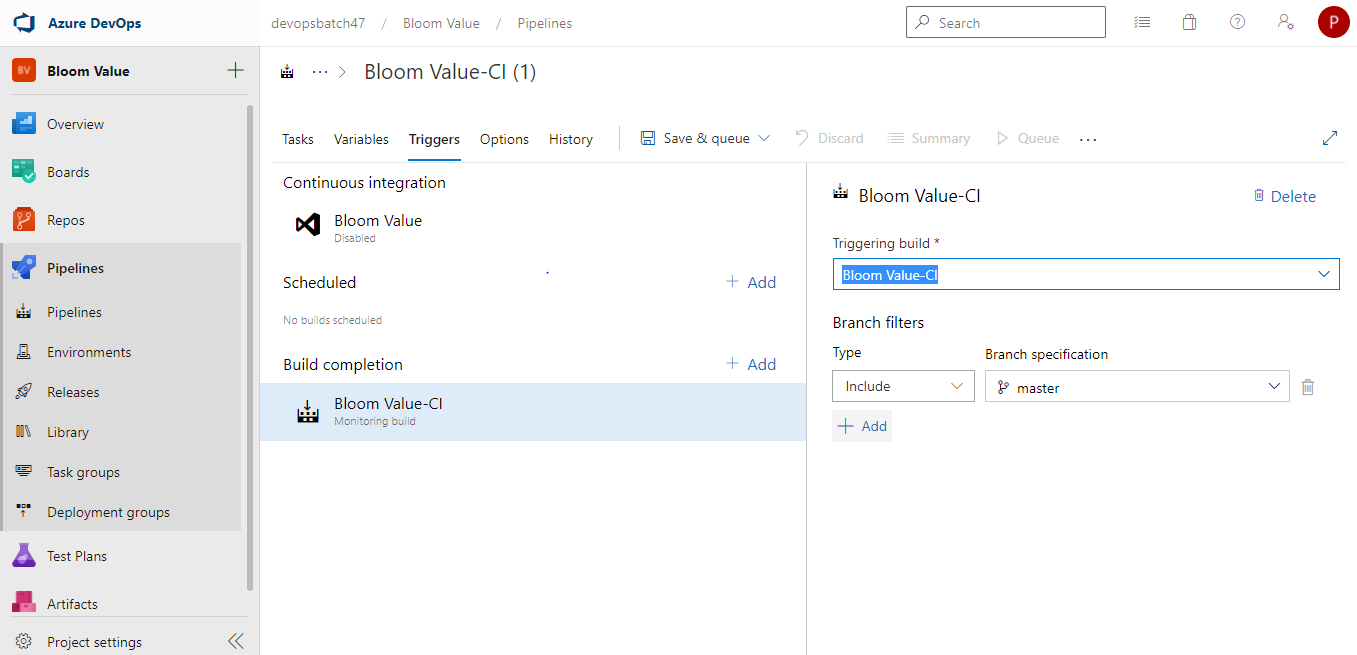
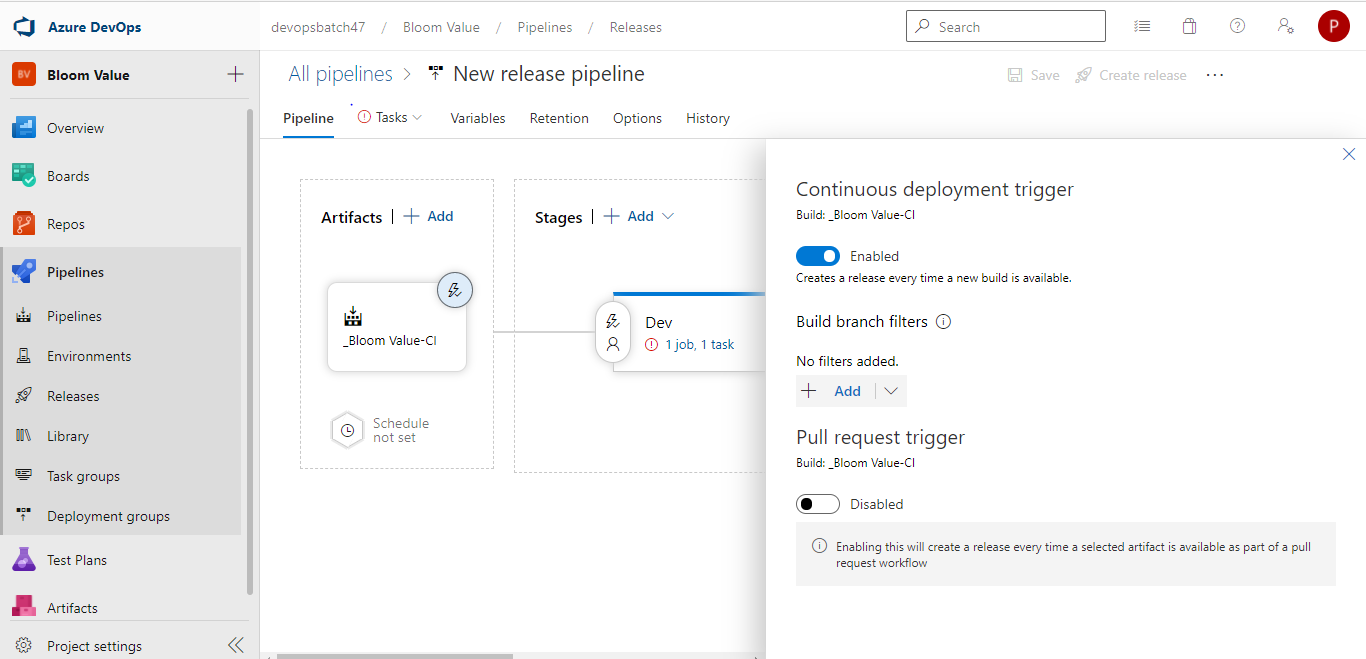
**Scenario – 1**

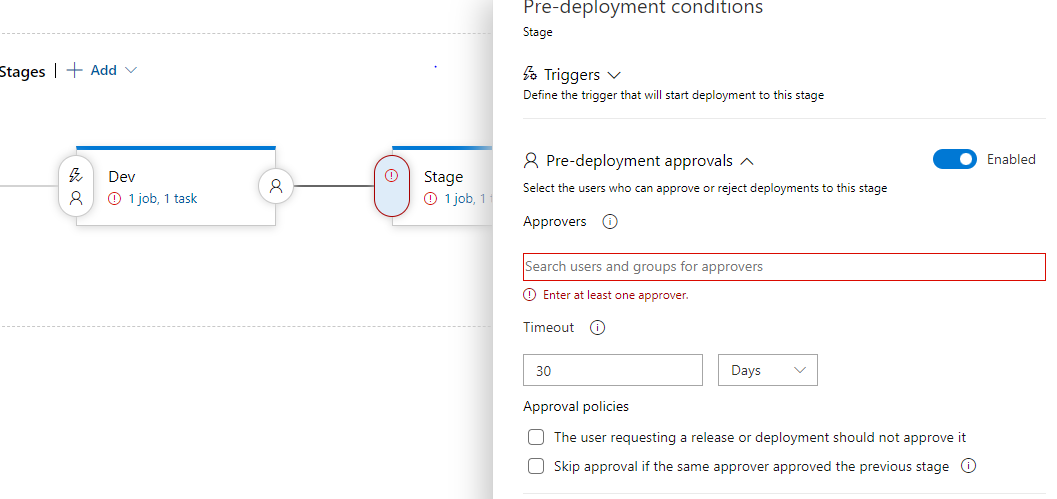
1. In azure devops when we are creating a build pipeline we can add some triggers to that pipeline like mentioning the specific git branch. So when ever the source code is checked in automatically the new build is created.



1. In the same pipeline there is a option under triggers tab called build completion. So, when we trigger a pipeline and that pipeline build completes then another pipeline starts running. As mentioned, if any build pipeline fails due to test case failure then other pipelines also not triggered.
2. 
3. In the release pipeline once we select the artifact and for each environment we add different stages. But in the dev environment we select the continuous deployment then the artifact created in the build pipeline is deployed to the dev environment.
4. As mentioned in the above point we can select the continuous deployment. So, the package can be deployed to the other environments in the release pipeline. Refer to below screenshot of how we can enable the continuous deployment once we add the artifacts on top, we can see thunder button which will be used to open the pop up as shown in screenshot.



1. We can add the approvals between each environment before deploying the package. Please refer below screenshot.



trigger:

 branches: master

pool:

  name: Azure Pipelines

  demands:

  - msbuild

  - visualstudio

  - vstest

steps:

- task: NuGetToolInstaller@0

  displayName: 'Use NuGet 4.4.1'

  inputs:

    versionSpec: 4.4.1

- task: NuGetCommand@2

  displayName: 'NuGet restore'

  inputs:

    restoreSolution: '$(Parameters.solution)'

- task: VSBuild@1

  displayName: 'Build solution \*\*\\*.sln'

  inputs:

    solution: '$(Parameters.solution)'

    msbuildArgs: '/p:DeployOnBuild=true /p:DeployDefaultTarget=WebPublish /p:WebPublishMethod=FileSystem /p:publishUrl="$(Agent.TempDirectory)\WebAppContent\\"'

    platform: '$(BuildPlatform)'

    configuration: '$(BuildConfiguration)'

- task: ArchiveFiles@2

  displayName: 'Archive Files'

  inputs:

    rootFolderOrFile: '$(Agent.TempDirectory)\WebAppContent'

    includeRootFolder: false

- task: VSTest@2

  displayName: 'VsTest - testAssemblies'

  inputs:

    testAssemblyVer2: |

     \*\*\$(BuildConfiguration)\\*test\*.dll

     !\*\*\obj\\*\*

    platform: '$(BuildPlatform)'

    configuration: '$(BuildConfiguration)'

- task: AzureRmWebAppDeployment@4

  inputs:

    ConnectionType: 'AzureRM'

    appType: 'webApp'

    WebAppName: 'dcbkzc'

    deployToSlotOrASE: true

    SlotName: 'production'

    packageForLinux: '$(System.DefaultWorkingDirectory)/\*\*/\*.zip'

- task: PublishSymbols@2

  displayName: 'Publish symbols path'

  inputs:

    SearchPattern: '\*\*\bin\\*\*\\*.pdb'

    PublishSymbols: false

  continueOnError: true

- task: PublishBuildArtifacts@1

  displayName: 'Publish Artifact: drop'

  inputs:

    PathtoPublish: '$(build.artifactstagingdirectory)'

  condition: succeededOrFailed()

**Scenario -2:**

1. We don’t create any artifacts as we are using the terraform script which will be picked up during the pipeline execution and executed automatically to create the infrastructure.

2. Firstly we will be using the source code management tool to store the terraform script and the list of SCM tools are like GIT,TFS,SVN,Bitbucket ….etc

3. To create this pipeline we should firstly need to install the terraform in agent then we will be executing the “terraform plan” command using the CLI task. Nextly once the init statement executed then we add the another CLI task to execute the “terraform apply” statement which will create the infrastructure as per the script.

4. provider "azurerm" {

features {}

}

resource "azurerm\_resource\_group" "rg" {

name = "my-resources"

location = "West Europe"

}

resource "azurerm\_network\_security\_group" "nsg" {

name = "nsg"

location = "${var.location}"

resource\_group\_name = "${azurerm\_resource\_group.resourceGroup.name}"

security\_rule {

name = "HTTPS"

priority = 1000

direction = "Inbound"

access = "Allow"

protocol = "Tcp"

source\_port\_range = "\*"

destination\_port\_range = "443"

source\_address\_prefix = "\*"

destination\_address\_prefix = "\*"

}

security\_rule {

name = "HTTP"

priority = 1010

direction = "Inbound"

access = "Allow"

protocol = "Tcp"

source\_port\_range = "\*"

destination\_port\_range = "80"

source\_address\_prefix = "\*"

destination\_address\_prefix = "\*"

}

}

resource "azurerm\_virtual\_network" "myvnet" {

name = "my-vnet"

address\_space = ["10.0.0.0/16"]

location = "northeurope"

resource\_group\_name = azurerm\_resource\_group.rg.name

}

resource "azurerm\_subnet" "frontendsubnet" {

name = "frontendSubnet"

resource\_group\_name = azurerm\_resource\_group.rg.name

virtual\_network\_name = azurerm\_virtual\_network.myvnet.name

address\_prefix = "10.0.1.0/24"

}

resource "azurerm\_subnet" "backendsubnet" {

name = "backendSubnet"

resource\_group\_name = azurerm\_resource\_group.rg.name

virtual\_network\_name = azurerm\_virtual\_network.myvnet.name

address\_prefix = "10.0.2.0/24"

}

resource "azurerm\_network\_interface" "main\_front\_NIC" {

name = "${var.prefix}-nic-front"

location = azurerm\_resource\_group.main.location

resource\_group\_name = azurerm\_resource\_group.main.name

ip\_configuration {

name = "testconfiguration1"

subnet\_id = azurerm\_subnet. frontendsubnet.id

private\_ip\_address\_allocation = "Dynamic"

}

}

data "azurerm\_key\_vault" "azvault" {

name = "MyKeyVault"

resource\_group\_name = "MyVaultRG"

}

data "azurerm\_key\_vault\_secret" "front\_secret" {

name = "FrontVMPassword"

key\_vault\_id = "${data.azurerm\_key\_vault.azvault.id}"

}

data "azurerm\_key\_vault\_secret" "back\_secret" {

name = "BackVMPassword"

key\_vault\_id = "${data.azurerm\_key\_vault.azvault.id}"

}

resource "azurerm\_virtual\_machine" "main\_front\_VM" {

name = "${var.prefix}-vm-front"

location = azurerm\_resource\_group.main.location

resource\_group\_name = azurerm\_resource\_group.main.name

network\_interface\_ids = [azurerm\_network\_interface.main\_front\_NIC.id]

vm\_size = "Standard\_DS1\_v2"

storage\_image\_reference {

publisher = "Canonical"

offer = "UbuntuServer"

sku = "16.04-LTS"

version = "latest"

}

storage\_os\_disk {

name = "myosdisk1"

caching = "ReadWrite"

create\_option = "FromImage"

managed\_disk\_type = "Standard\_LRS"

}

os\_profile {

computer\_name = "hostname"

admin\_username = "testadmin"

admin\_password = "${data.azurerm\_key\_vault\_secret.front\_secret.value}" }

os\_profile\_linux\_config {

disable\_password\_authentication = false

}

}

resource "azurerm\_network\_interface" "main\_back\_NIC" {

name = "${var.prefix}-nic-back"

location = azurerm\_resource\_group.main.location

resource\_group\_name = azurerm\_resource\_group.main.name

ip\_configuration {

name = "testconfiguration1"

subnet\_id = azurerm\_subnet.backendsubnet.id

private\_ip\_address\_allocation = "Dynamic"

}

}

resource "azurerm\_virtual\_machine" "main\_back\_VM" {

name = "${var.prefix}-vm-back"

location = azurerm\_resource\_group.main.location

resource\_group\_name = azurerm\_resource\_group.main.name

network\_interface\_ids = [azurerm\_network\_interface.main\_front\_NIC.id]

vm\_size = "Standard\_DS1\_v2"

storage\_image\_reference {

publisher = "Canonical"

offer = "UbuntuServer"

sku = "16.04-LTS"

version = "latest"

}

storage\_os\_disk {

name = "myosdisk1"

caching = "ReadWrite"

create\_option = "FromImage"

managed\_disk\_type = "Standard\_LRS"

}

os\_profile {

computer\_name = "hostname"

admin\_username = "testadmin"

admin\_password = "${data.azurerm\_key\_vault\_secret.back\_secret.value}"

}

os\_profile\_linux\_config {

disable\_password\_authentication = false

}

}

resource "azurerm\_storage\_account" "example" {

name = "storageaccountname"

resource\_group\_name = azurerm\_resource\_group.example.name

location = azurerm\_resource\_group.example.location

account\_tier = "Standard"

account\_replication\_type = "GRS"

}

5. (I).we can add the secrets name to the library under azure pipelines in azure devops. Then those pipeline variables can be used during the pipeline execution to fetch the secrets from the key vault task in the pipeline and get the secret and store it as the environment variables in the agent we run our pipeline so that the terraform script can pick the secret while execution.

(II).In the terraform script itself we can add the data blocks which will access the keyvault using terraform and use the passwords fetched from the keyvault directly to the terraform scripts.

Data “azurerm\_key\_vault” “azvault” {

Name = “MyKeyVault”

Resource\_group\_name = “MyVaultRG”

}

Data “azurerm\_key\_vault\_secret” “front\_secret” {

Name = “FrontVMPassword”

Key\_vault\_id = “${data.azurerm\_key\_vault.azvault.id}”

}

Data “azurerm\_key\_vault\_secret” “back\_secret” {

Name = “BackVMPassword”

Key\_vault\_id = “${data.azurerm\_key\_vault.azvault.id}”

}