Continuous Integration using YAML Pipelines

- Introduction to YAML Pipeline
- Building Azure DevOps Pipeline using YAML
- Publishing results to Artifacts
- Triggering Continuous Integration in YAML
- Filtering Tasks based on branch being built
- Using Templates to Build Multiple Configurations
- Build on Multi-Platform pipeline

Introduction to YAML Pipeline

- **YAML**: **Y**et **A**nother **M**arkup **L**anguage
- The pipeline is versioned with your code and follows the same branching structure. You get validation of your changes through code reviews in pull requests and branch build policies.
- Every branch you use can modify the build policy by modifying the **azure-pipelines.yml** file.
- A change to the build process might cause a break or result in an unexpected outcome. Because the change is in version control with the rest of your codebase, you can more easily identify the issue.

Exercise: Build Azure DevOps Pipeline using YAML

1. Pipeline → **New Pipeline**
2. Select Tab → Select **HelloWorldApp.Web** repository.
3. On the **Configure** tab, select **ASP.NET Core** (click on choose more)
4. In Visual Studio Code, modify ***azure-pipelines.yml*** as you see here:

```
trigger:
- 'none'

pool:
  vmImage: 'ubuntu-16.04'

steps:
- task: DotNetCoreCLI@2
  displayName: 'Restore project dependencies'
  inputs:
    command: 'restore'
    projects: '**/*.csproj'
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

```
- task: DotNetCoreCLI@2
  displayName: 'Build the project - Release'
  inputs:
    command: 'build'
    arguments: '--no-restore --configuration Release'
    projects: '**/*.csproj'

- task: DotNetCoreCLI@2
  displayName: 'Publish the project - Release'
  inputs:
    command: 'publish'
    projects: '**/*.csproj'
    publishWebProjects: false
    arguments: '--no-build --configuration Release --output $(Build.ArtifactStagingDirectory)/Release'
    zipAfterPublish: true

- task: PublishBuildArtifacts@1
  displayName: 'Publish Artifact: drop'
  condition: succeeded()
```

The **first task** uses the **DotNetCoreCLI@2** task to ***publish***, or package, the application's build results (including its dependencies) into a folder.

The **zipAfterPublish** argument specifies to add the built results to a .zip file.

The **second task** uses the PublishBuildArtifacts@1 task to publish the .zip file to Azure Pipelines.

Triggering Continuous Integration in YAML

You can control which branches get CI triggers with a simple syntax:

```
trigger:
- master
- releases/*
```

```
# A pipeline with no CI trigger
trigger: none
```

```
# specific branch build with batching
trigger:
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

```
batch: true
branches:
  include:
  - master
```

specific branch build

```
trigger:
 branches:
  include:
  - master
  - releases/*
  exclude:
  - releases/old*
 paths:
  include:
  - docs/*
  exclude:
  - docs/README.md
```

In addition to specifying branch names in the branches lists, you can also configure triggers based on tags by using the following format:

```
trigger:
 branches:
  include:
    refs/tags/{tagname}
  exclude:
    refs/tags/{othertagname}
```

You can specify the target branches for your pull request builds.

YAML PR triggers are only supported in GitHub and Bitbucket Cloud and for NOT for Azure Repos

```
pr:
- master
- releases/*
```

If **no pr triggers** appear in your YAML file, pull request builds are **automatically enabled** for all branches, as if you

```
pr:
 branches:
  include:
  - '*'  # must quote since "*" is a YAML reserved character; we want a string
```

Override YAML triggers

PR and CI triggers that are configured in YAML pipelines can be overridden in the pipeline settings, and by default, new pipelines automatically override YAML PR triggers. To configure this setting, select **Triggers** from the settings menu while editing your YAML pipeline.

Using Templates to Build Multiple Configurations

A *template* enables you to define common build tasks one time and reuse those tasks multiple times.

You call a template from the parent pipeline as a build step. You can pass parameters into a template from the parent pipeline.

Templates combine the content of multiple YAML files into a single pipeline.

Requirement: We need to now repeat the two tasks **Build** and **Publish** but replace **Release** with **Debug**

1. Create a *templates* directory at the root of your project:
2. Create a new File **templates\build.yml**

```
parameters:
   buildConfiguration: 'Release'

steps:
- task: DotNetCoreCLI@2
 displayName: 'Build the project - ${{ parameters.buildConfiguration }}'
 inputs:
   command: 'build'
   arguments: '--no-restore --configuration ${{ parameters.buildConfiguration }}'
   projects: '**/*.csproj'

- task: DotNetCoreCLI@2
 displayName: 'Publish the project - ${{ parameters.buildConfiguration }}'
 inputs:
   command: 'publish'
   projects: '**/*.csproj'
   publishWebProjects: false
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

```
  arguments: '--no-build --configuration ${{ parameters.buildConfiguration }} --
output $(Build.ArtifactStagingDirectory)/${{ parameters.BuildConfiguration }}'
  zipAfterPublish: true
```

Note the two differences

- In a template file, you use the parameters section instead of variables to define inputs.
- In a template file, you use ${{ }} syntax instead of $() to read a parameter's value. When you read a parameter's value, you include the parameters section in its name. For example, ${{ parameters.buildConfiguration }}.

3. Call the template from the pipeline
4. Place the below two task just above the last task in **azure-pipelines.yml**

```
- template: templates/build.yml
 parameters:
  buildConfiguration: 'Debug'

- template: templates/build.yml
 parameters:
  buildConfiguration: 'Release'
```

5. You see that the pipeline produces a *.zip* file for both the **Debug** configuration and the **Release** configuration.

Job templates:

In this example, a single job is repeated on three platforms. The job itself is specified only once.

 # File: jobs/build.yml

```
parameters:
        name: ''
        pool: ''
        sign: false
jobs:
        - job: ${{ parameters.name }}
          pool: ${{ parameters.pool }}
          steps: - script: npm install
          - script: npm test
          - ${{ if eq(parameters.sign, 'true') }}:
                - script: sign
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

```
# File: azure-pipelines.yml
jobs:
    - template: jobs/build.yml # Template reference
      parameters:
          name: macOS
          pool:
              vmImage: 'macOS-10.14'

    - template: jobs/build.yml # Template reference
      parameters:
          name: Linux
          pool:
              vmImage: 'ubuntu-16.04'
    - template: jobs/build.yml # Template reference
      parameters:
          name: Windows
          pool:
              vmImage: 'ubuntu-16.04'
          sign: true #Extra step on windows only
```

Understanding YAML File Format

YAML file Format:

```
name: string  # build numbering format
resources:
  pipelines: [ pipelineResource ]
  containers: [ containerResource ]
  repositories: [ repositoryResource ]
variables: { string: string } | [ variable | templateReference ]
trigger: trigger
pr: pr
stages: [ stage | templateReference ]
```

- If you have a single stage, you can omit stages and directly specify jobs:
- If you have a single stage and a single job, you can omit those keywords and directly specify steps:

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Example:

```
name: $(Date:yyyyMMdd)
variables:
 var1: value1
jobs:
- job: One
 steps:
 - script: echo First step!
```

Stage

A stage is a logical boundary in the pipeline. It can be used to mark separation of concerns (e.g., Build, QA, and production). Each stage contains **one or more jobs**.

By default, stages run sequentially, starting only after the stage ahead of them has completed. You can manually control when a stage should run using approval checks.

Example:

```
stages:
- stage: Build
 jobs:
 - job: BuildJob
  steps:
  - script: "echo Building!"
- stage: Test
 jobs:
 - job: TestOnWindows
  steps:
  - script: echo Testing on Windows!
 - job: TestOnLinux
  steps:
  - script: echo Testing on Linux!
- stage: Deploy
 jobs:
 - job: Deploy
  steps:
  - script: echo Deploying the code!
  - script: echo Deploying the code!
```

Job

A job is a collection of **linear series of steps** to be run by an agent or on the server. It's a units of work assignable to a particular machine. More than one jobs can run in parallel.

```
jobs:
- job: MyJob
  displayName: My First Job
  continueOnError: true
  workspace:
    clean: outputs
  steps:
  - script: echo My first job
```

Step

A step is the smallest building block of a pipeline. A step can either be a **script or a task**. For example, a pipeline might consist of build and test steps.

Tasks

A task is simply a pre-created script offered as a convenience to you. This abstraction makes it easier to run common build functions.

```
steps:
- script: echo This runs in the default shell on any machine
- bash: |
    echo This multiline script always runs in Bash.
    echo Even on Windows machines!
- pwsh: |
    Write-Host "This multiline script always runs in PowerShell Core."
    Write-Host "Even on non-Windows machines!"
- task: DotNetCoreCLI@2
    displayName: 'Build the project'
    inputs:
      command: 'build'
      arguments: '--no-restore --configuration Release'
      projects: '**/*.csproj'
```