How to start with an Azure Pipeline?

> *The below are some general tasks which can be included in a pipeline. The Pipeline design depends upon the project requirement. It can be customized as required using different tasks and tools.*

Step-by-step guide followed by some screenshots

1. Navigate to Pipelines > Pipelines > Your Build Pipeline
2. Click Edit
3. Use one of the two Abbvie Agent Pools (windows or Ubuntu) as your build agent
    a. Ubuntu: 'abv-linux-agent-pool-latest'
    b. Windows: 'BTS-DEVOPS-WIN-PROD'/'BTS-DEVOPS-WIN-DEV'
4. Add the Prepare Analysis Configuration step to your pipeline before the build stage
    a. Choose the EnterpriseServiceConnection service connection
    b. Choose you build type
    c. Advanced Section
        i. Add sonar.projectKey with your project key to link sonar correctly to your project
            1. sonar.projectKey=helloworld
        ii. Add sonar.branch.name=$(Build.SourceBranchName)
            1. This will link your sonar scan to the current branch that you are running on
        iii. Add any other additional properties as required
5. Configure run analysis step
    a. If you chose 'Use standalone scanner' in step 4
        i. Add the Run Code Analysis stage to your build after your build stage
    b. If you chose another option
        i. Check the code analysis box in your build step to ensure it runs
6. *Optional* : After the analysis, add the 'Publish Quality Gate Result" step
    a. Displays sonar status results with build in Azure DevOps
        i. To view results published this way, go to the Extensions tab on the build
7. Add the SonarQube build breaker step
    a. Will fail your pipeline if the Quality Gate fails
8. If using a linux agent
    a. Click on Variables
    b. Click the +
    c. Add the following variable
        i. Name: NODE_EXTRA_CA_CERTS
        ii. Value: /etc/ssl/certs/tls-ca-bundle.pem
        iii. This ensures that the sonar tasks know where to find the AbbVie SSL certs
9. Save

Screenshots of Pipeline Tasks:

Triggers and Pool:

- In order to enable CI in a pipeline, we can have the triggers. Whenever there are commits in the branches mentioned under triggers, the pipeline automatically runs.
- The pipeline can also be triggered manually. For this, you have to disable the CI. Follow the link to know how to do it.
- Select the desired pool according to the requirement.

```yaml
trigger:
  branches:
    include:
    - develop
    - qa/abbvie-ous-abbvieid-eapi-v1
    - uat/abbvie-ous-abbvieid-eapi-v1

  paths:
    include:
    - 'abbvie-ous-abbvieid-eapi-v1/'

pool:
  name: 'vmss-ubu-1804-agentpool'
```

Stages & Tasks:

- A Pipeline can have no. of stages like Build, Deploy, Merge and so on... A stage contains jobs, Job contains steps, A step contains Tasks/ Scripts. The above task is to Download Secure File. A secure file can be uploaded to Library and encrypted. Specify the name of the secure file uploaded to download in the pipeline.

```yaml
stages:
- stage: Build
  displayName: Build

  jobs:
  - job: MavenBuild
    displayName: Maven Build

    steps:
    - task: DownloadSecureFile@1
      name: settingsXml
      displayName: 'Download settings.xml from pipeline library secure file'
      inputs:
        secureFile: 'ous-mule-settings.xml'
```

Certificate Paths:

- The above screenshots shows how to specify Certificate path for any tool in the pipeline. This path refers to the SonarQube Certificate

```yaml
- name: NODE_EXTRA_CA_CERTS
  value: /etc/ssl/certs/tls-ca-bundle.pem
```

SonarQube Prepare Analysis & Publish:

- If your pipeline needs the SonarQube Analysis for code scan, include the above two tasks. One for SonarQube Analysis Preparation and the other to publish Sonar Scan Results in the pipeline output.
- Between these two tasks the below Maven Build Task has to be placed.

```yaml
- task: SonarQubePrepare@4
  inputs:
    SonarQube: 'EnterpriseServiceConnection'
    scannerMode: 'Other'
    extraProperties: |
      sonar.projectKey=$(api-folder-name)
      sonar.branch.name=$(Build.SourceBranchName)
      sonar.sources=src
```

```yaml
- task: SonarQubePublish@4
  inputs:
    pollingTimeoutSec: '300'

- task: sonar-buildbreaker@8
  inputs:
    SonarQube: 'EnterpriseServiceConnection'
```

Build Task:

- This task is the heart of any pipeline. Maven Build. This is one example of Maven Build Task with necessary steps. If needed we can specify code coverage options also. Azure DevOps supports many Build tools like Ant, Gradle. Can choose anyone based on the project specifications. Link for YML Snippet.

```yaml
- task: Maven@3
  displayName: 'Clean Test and Build'
  inputs:
    mavenPomFile: '$(api-folder-name)/pom.xml'
    goals: 'clean test package'
    options: '-s $(settingsXml.secureFilePath) -Dversion.suffix=$(artifact.suffix)'
    publishJUnitResults: true
    jdkVersionOption: 'default' #Java 8 is the default in the BTS agent
    javaHomeOption: 'JDKVersion'
    mavenVersionOption: 'Path'
    mavenSetM2Home: false
    mavenDirectory: '/usr/share/maven'
    mavenFeedAuthenticate: true
    sonarQubeRunAnalysis: true
    sqMavenPluginVersionChoice: 'latest'
    findBugsRunAnalysis: true
```

Copy Files Task:

- Copy files task helps us to copy files based on the path mentioned in the inputs. Can use wildcards to specify the path of the packages/files.
- Upload task - As the name says, it uploads the copied files to the Pipeline Build(Current).

```yaml
- task: CopyFiles@2
  displayName: 'Copy JAR to Artifact Staging Directory'
  inputs:
    SourceFolder: '$(System.DefaultWorkingDirectory)'
    Contents: '**/target/*.?(war|jar)'
    TargetFolder: $(Build.ArtifactStagingDirectory)

- upload: $(Build.ArtifactStagingDirectory)
  artifact: jardrop-$(pomversion)
```

Deploy Task:

- This Maven Deploy Stage has the steps included downloading the settings.xml file, Artifact from the Current Build and to print the pom version before the Deploy to Target Environment happens.

```yaml
- stage: Deploy
  displayName: Deploy
  dependsOn: Build
  condition: succeeded()

  jobs:
  - deployment: MavenDeploy
    displayName: Maven Deploy
    environment: $(azure.environment)
    variables:
      pomversion: $[stageDependencies.Build.MavenBuild.outputs['getPomProperties.pomversionoutput']]
    strategy:
      runOnce:
        deploy:
          steps:
          - task: DownloadSecureFile@1
            name: settingsXml
            displayName: 'Download settings.xml from pipeline library secure file'
            inputs:
              secureFile: 'ous-mule-settings.xml'

          - script: |
              echo pomVersionFromStage: $(pomversion)
            displayName: 'Show pom artifacts variables'

          - task: DownloadPipelineArtifact@2
            inputs:
              source: current
```

Deploy to PROD:

- This is where the package deploys to the target environment through Maven Task. Here in this scenario, The target is Cloud Hub.

```yaml
- task: Maven@3
  displayName: 'Deploy to Cloudhub...'
  inputs:
    mavenPomFile: '$(Pipeline.Workspace)/pomdrop-$(pomversion)/$(api-folder-name)/pom.xml'
    goals: 'mule:deploy'
    options: '-s $(settingsXml.secureFilePath) -DskipTests -DmuleDeploy -Dversion.suffix=$(artifact.suffix)
    publishJUnitResults: false
    javaHomeOption: 'JDKVersion'
    jdkVersionOption: 'default' #Java 8 is the default in the BTS agent
    mavenVersionOption: 'Path'
    mavenSetM2Home: false
    mavenDirectory: '/usr/share/maven'
    mavenFeedAuthenticate: true
    sonarQubeRunAnalysis: false
```

Merge to next branch:

- This is a Merge stage where the pipeline can merge to the next branch.
- Gated environments can be enabled here to have a manual intervention to approve the code merge to the next branch.

```
- ${{ if eq(variables['build.SourceBranchName'], 'develop') }}:
  - stage: MergeQA
    displayName: Merge To QA
    dependsOn: Deploy
    condition: succeeded()

    jobs:
    - deployment: GitMerge
      displayName: Git Merge
      environment: 'GIT-MERGE'
      strategy:
        runOnce:
          deploy:
            steps:
            - checkout: self
              clean: true
              persistCredentials: true

            - task: Bash@3
              displayName: 'Merge to QA'
              inputs:
                targetType: 'inline'
                script: |
                  echo "SET GIT CONFIG"
                  git config --global user.email "serviceuser@abbvie.com"
                  git config --global user.name "Service User" #should change to a service account

                  echo $(Build.SourceBranchName)
                  echo $(Build.SourceBranch)

                  echo 'develop branch'
                  git checkout qa/abbvie-ous-abbvieid-eapi-v1
                  echo "GIT MERGE TO QA"
                  git merge origin/develop -m "Merge to qa/abbvie-ous-abbvieid-eapi-v1"
                  git push origin
                  git checkout develop
```

Related articles

- How to add users to AD security groups for access to DevOps Tools?
- How to integrate a SonarQube Project to Azure Pipeline?
- How to create a Change Request (CR) Release pipeline for Production Release?
- How to start with an Azure Pipeline?
- How to access Azure DevOps?