parameters:

target\_salesforce\_org\_branch: 'NOT\_DEFINE'

secure\_file: 'NOT\_DEFINE'

environment\_name: 'NOT\_DEFINE'

steps:

- task: Bash@3

displayName: Send Build Start notification

continueOnError: true

inputs:

filePath: './azure-devops-shared-automation/devops-scripts/utilities.sh'

workingDirectory: '$(Build.SourcesDirectory)/azure-devops-shared-automation/devops-scripts'

failOnStderr: true

arguments: google\_notification "$(googleChatChannelWebhookUrl)" "\*Started $BUILD\_REASON BUILD $(Build.BuildId)\* from $(Build.SourceBranch) of $(Build.Repository.Name) in ${{ parameters.environment\_name }} environment. The build was triggered by $(Build.RequestedFor)." "$(Build.SourcesDirectory)"

- bash: |

export V\_SOURCE\_BRANCH=$(echo $(Build.SourceBranch) | sed -e "s/refs\/heads\///g")

echo "##vso[task.setvariable variable=SOURCE\_BRANCH;isOutput=true]$V\_SOURCE\_BRANCH"

echo "Source (current) branch: $V\_SOURCE\_BRANCH"

export V\_REPO\_NAME=$(echo $(echo $(Build.Repository.Name) | cut -f2 -d"/"))

echo "##vso[task.setvariable variable=REPOSITORY\_NAME;isOutput=true]$V\_REPO\_NAME"

echo "Repo name: $V\_REPO\_NAME"

echo "TESTLEVEL: $(TESTLEVEL)"

if ! [[ "$(TESTLEVEL)" =~ ^(NoTestRun|RunLocalTests|RunSpecifiedTests)$ ]]; then

echo "\*\*\* Unsupported TESTLEVEL value found, Re-run providing supported value as 'NoTestRun' or 'RunLocalTests' or 'RunSpecifiedTests' ...failing the pipeline \*\*\*"

exit 1

fi

if [[ ("$(TESTLEVEL)" == 'RunSpecifiedTests') && ( -z "$(RUNTESTS)") ]]; then

echo "\*\*\* Please specify Test Class name(s) in RUNTESTS variable for TESTLEVEL=RunSpecifiedTests and try again ...failing the pipeline \*\*\*"

exit 1

elif [[ ("$(TESTLEVEL)" == 'RunSpecifiedTests') && ! ( -z "$(RUNTESTS)") ]]; then

echo "TESTLEVEL: $(TESTLEVEL)"

echo "RUNTESTS: $(RUNTESTS)"

fi

displayName: 'Set Variables'

name: set\_vars

- bash: |

echo "Check Source Branch Name; Confluence link: https://woolworths-agile.atlassian.net/wiki/spaces/RM/pages/32522520646/Naming+Convention+for+Git+Feature+Branch+proposed"

echo "REPOSITORY\_NAME: $(set\_vars.REPOSITORY\_NAME)"

echo "PR\_BRANCH: $(set\_vars.SOURCE\_BRANCH)"

echo "Pull Request Target Branch: $(System.PullRequest.TargetBranch)"

echo "Pull Request Source Branch: $(System.PullRequest.SourceBranch)"

echo "Pull Requst Number: $(System.PullRequest.PullRequestNumber)"

export TEAM=$(echo $(System.PullRequest.SourceBranch) | cut -d- -f1)

export TEAM\_LENGTH=${#TEAM}

if ! [[ $(System.PullRequest.SourceBranch) == Org\_monitoring\_\* ]]; then

if [[ $(System.PullRequest.SourceBranch) =~ [A-Z] ]]; then

echo "PR Source Branch Name Violation: Not allowed Uppercase letter in Source branch name... failing pipeline"

echo "PR Source Branch Name: $(System.PullRequest.SourceBranch)"

exit 1

fi

if ! [[ $(System.PullRequest.SourceBranch) == \*-\* ]]; then

echo "PR Source Branch Name Violation: Not found hyphen after first word (TEAM name is considered as first word), branch name cannot have only TEAM name ... failing pipeline"

echo "PR Source Branch Name: $(System.PullRequest.SourceBranch)"

exit 1

fi

if [[ $TEAM == \*[\_/.]\* ]]; then

echo "Team name: $TEAM"

echo "PR Source Branch Name Violation: Not allowed special charactor (underscore or forward slash or dot) in TEAM name (first word before hypen in Source branch name)"

echo "PR Source Branch Name: $(System.PullRequest.SourceBranch)"

exit 1

fi

if [[ $TEAM\_LENGTH -lt 3 || $TEAM\_LENGTH -gt 10 ]]; then

echo "Team name: $TEAM"

echo "PR Source Branch Name Violation: First word before hyphen cannot be less than 3 charactor and more than 10 (TEAM name is considered as first word)... failing pipeline"

echo "PR Source Branch Name: $(System.PullRequest.SourceBranch)"

exit 1

fi

if ! [[ $TEAM =~ [a-z] ]]; then

echo "Team name: $TEAM"

echo "PR Source Branch Name Violation: First word before hyphen cannot have only numbers... failing pipeline"

echo "PR Source Branch Name: $(System.PullRequest.SourceBranch)"

exit 1

fi

echo "TEAM: $TEAM"

fi

displayName: '==> Check Source Branch Name'

condition: and(eq(variables['Build.Reason'], 'PullRequest'), or(eq(variables['Build.Repository.Name'], 'woolworthslimited/Partnerhub'),eq(variables['Build.Repository.Name'], 'woolworthslimited/wow-bunch')))

- bash: |

echo "$(set\_vars.REPOSITORY\_NAME)"

echo "$(set\_vars.SOURCE\_BRANCH)"

mkdir -p $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/pmd\_result\_$(Build.BuildId)

mkdir -p $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)

mkdir -p $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/apex\_test\_output\_$(Build.BuildId)

mkdir -p $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/apex\_cov\_output\_$(Build.BuildId)

mkdir -p $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/target\_sf\_org\_$(Build.BuildId)

mkdir -p $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/tmp1

ls -ltr $(System.DefaultWorkingDirectory)/\*/\*\*

displayName: 'Create output directories'

# Check current CLI version

- bash: |

sfdx --version

sf --version

displayName: 'Check sfdx CLI version'

# Download Key

- task: DownloadSecureFile@1

name: jwtKeyFile

displayName: 'Download key'

inputs:

secureFile: ${{ parameters.secure\_file }}

# Login to Salesforce Org

- script: |

sf org login jwt --client-id $(salesforceClientid) --jwt-key-file $(jwtKeyFile.secureFilePath) --username $(salesforceUsername) --alias $(alias)

displayName: 'Authenticate to Salesforce org'

# Install plugins

- bash: |

#export NODE\_OPTIONS=--max\_old\_space\_size=8192

echo 'y' | sfdx plugins:install sfdx-git-delta

echo 'y' | sfdx plugins:install @salesforce/sfdx-scanner@3.19.0

echo 'List plugins installed'

sfdx plugins

displayName: 'Install plugins'

# Generate Delta Package - diff branches

- bash: |

echo $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)

cd $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)

export SOURCE\_API\_VERSION=$(cat sfdx-project.json | jq -r .sourceApiVersion)

echo "In sfdx-project.json sourceApiVersion=$SOURCE\_API\_VERSION"

export LAST\_DEPLOYED\_COMMIT=$(git show-ref -s refs/remotes/origin/${{ parameters.target\_salesforce\_org\_branch }})

sfdx sgd:source:delta -t HEAD -f $LAST\_DEPLOYED\_COMMIT --output "$(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)" --generate-delta --api-version=$SOURCE\_API\_VERSION

displayName: 'Generate Delta Package - diff branches'

condition: ne(variables['build.sourceBranch'], 'refs/heads/${{ parameters.target\_salesforce\_org\_branch }}')

# Generate Delta Package - Self

- bash: |

echo $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)

cd $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)

export SOURCE\_API\_VERSION=$(cat sfdx-project.json | jq -r .sourceApiVersion)

echo "In sfdx-project.json sourceApiVersion=$SOURCE\_API\_VERSION"

sfdx sgd:source:delta --to "HEAD" --from "HEAD^" --output "$(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)" --generate-delta --api-version=$SOURCE\_API\_VERSION

displayName: 'Generate Delta Package - Self'

condition: eq(variables['build.sourceBranch'], 'refs/heads/${{ parameters.target\_salesforce\_org\_branch }}')

# copy sfdx config and forceignore

- bash: |

cd $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)

cp ./sfdx-project.json ./artifact\_$(Build.BuildId)

cp ./.forceignore ./artifact\_$(Build.BuildId)

displayName: 'copy sfdx config and forceignore'

# Archive files

- task: ArchiveFiles@2

inputs:

rootFolderOrFile: '$(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)'

includeRootFolder: true

archiveType: 'zip'

tarCompression: 'none'

archiveFile: '$(Build.ArtifactStagingDirectory)/$(Build.BuildId).zip'

verbose: 'true'

- task: ArtifactoryGenericUpload@2

inputs:

artifactoryService: 'ado-jfrog'

specSource: 'taskConfiguration'

fileSpec: |

{

"files": [

{

"pattern": "$(Build.ArtifactStagingDirectory)/$(Build.BuildId).zip",

"target": "$(jfrogArtifactoryRepository)",

"flat": "true",

"recursive": "true",

"explode": "true"

}

]

}

collectBuildInfo: true

buildName: "$(jfrogArtifactoryRepository)"

buildNumber: '$(Build.BuildId)'

includeEnvVars: true

failNoOp: true

- task: ArtifactoryPublishBuildInfo@1

inputs:

artifactoryService: 'ado-jfrog'

buildName: "$(jfrogArtifactoryRepository)"

buildNumber: '$(Build.BuildId)'

continueOnError: true

- task: ArtifactoryXrayScan@1

inputs:

artifactoryService: 'ado-jfrog'

buildName: '$(jfrogArtifactoryRepository)'

buildNumber: '$(Build.BuildId)'

allowFailBuild: false

continueOnError: true

- task: ArtifactoryDiscardBuilds@1

inputs:

artifactoryService: 'ado-jfrog'

buildName: "$(jfrogArtifactoryRepository)"

maxDays: '30'

maxBuilds: '90'

deleteArtifacts: true

continueOnError: true

# PMD on Delta Pkg: Static Code Analysis

- script: |

cd $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)

echo "Running PMD on DELTA Pkg"

cp ./sfdx-project.json ./artifact\_$(Build.BuildId)

cp ./.forceignore ./artifact\_$(Build.BuildId)

if [[ -e rulesets.xml ]]; then

echo "Found rulesets.xml"

sfdx scanner:run --format=csv --target "$(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)/\*\*/default/classes/\*\*" --loglevel info --verbose --pmdconfig rulesets.xml -o $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/pmd\_result\_$(Build.BuildId)/delta\_pmd\_output\_$(Build.BuildId).csv

else

echo "Not found rulesets.xml"

sfdx scanner:run --format=csv --target "$(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)/\*\*/default/classes/\*\*" --loglevel info --verbose -o $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/pmd\_result\_$(Build.BuildId)/delta\_pmd\_output\_$(Build.BuildId).csv

fi

continueOnError: true

displayName: 'PMD: (DELTA) Deployment Pkg'

- bash: |

cd $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)

git worktree add $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/target\_sf\_org\_$(Build.BuildId) ${{ parameters.target\_salesforce\_org\_branch }}

git worktree list

displayName: 'Checkout TARGET SF ORG branch'

# PMD on entire TARGET SF ORG branch code: Static Code Analysis

- script: |

cd $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)

echo "Running PMD on branch: ${{ parameters.target\_salesforce\_org\_branch }}"

if [[ -e $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/target\_sf\_org\_$(Build.BuildId)/rulesets.xml ]]; then

echo "Found rulesets.xml"

sfdx scanner:run --format=csv --target "$(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/target\_sf\_org\_$(Build.BuildId)/\*\*/default/classes/\*\*" --loglevel info --verbose --pmdconfig $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/target\_sf\_org\_$(Build.BuildId)/rulesets.xml -o $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/pmd\_result\_$(Build.BuildId)/target\_salesforce\_org\_branch\_pmd\_output\_$(Build.BuildId).csv

else

echo "Not found rulesets.xml"

sfdx scanner:run --format=csv --target "$(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/target\_sf\_org\_$(Build.BuildId)/\*\*/default/classes/\*\*" --loglevel info --verbose -o $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/pmd\_result\_$(Build.BuildId)/target\_salesforce\_org\_branch\_pmd\_output\_$(Build.BuildId).csv

fi

continueOnError: true

displayName: 'PMD: (TARGET) SF ORG branch'

- bash: |

cd $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)

sed -e 's/artifact\_$(Build.BuildId)//g' $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/pmd\_result\_$(Build.BuildId)/delta\_pmd\_output\_$(Build.BuildId).csv | awk -F "\"\*,\"\*" '{print $3}' | egrep -v 'File' | sort > $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/pmd\_result\_$(Build.BuildId)/delta\_pmd\_output.csv

sed -e 's/target\_sf\_org\_$(Build.BuildId)//g' $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/pmd\_result\_$(Build.BuildId)/target\_salesforce\_org\_branch\_pmd\_output\_$(Build.BuildId).csv | awk -F "\"\*,\"\*" '{print $3}' | egrep -v 'File' | sort > $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/pmd\_result\_$(Build.BuildId)/target\_salesforce\_org\_branch\_pmd\_output.csv

export countProblemsInDelta=$(cat $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/pmd\_result\_$(Build.BuildId)/delta\_pmd\_output.csv | wc -l)

echo "Count of Problems in Delta Pkg: $countProblemsInDelta"

echo "List of Problems in Delta Pkg:"

cat $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/pmd\_result\_$(Build.BuildId)/delta\_pmd\_output.csv | uniq -c | tee $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/pmd\_result\_$(Build.BuildId)/count\_component\_delta\_pmd\_output.txt

export countProblemsInTarget=$(cat $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/pmd\_result\_$(Build.BuildId)/target\_salesforce\_org\_branch\_pmd\_output.csv | wc -l)

echo "Count of Problems in Target SF Org branch: $countProblemsInTarget"

echo "List of Problems in Target:"

cat $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/pmd\_result\_$(Build.BuildId)/target\_salesforce\_org\_branch\_pmd\_output.csv | uniq -c | tee $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/pmd\_result\_$(Build.BuildId)/count\_component\_target\_salesforce\_org\_branch\_pmd\_output.txt

comm -1 -3 <(sort $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/pmd\_result\_$(Build.BuildId)/target\_salesforce\_org\_branch\_pmd\_output.csv) <(sort $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/pmd\_result\_$(Build.BuildId)/delta\_pmd\_output.csv) | tee $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/pmd\_result\_$(Build.BuildId)/list\_new\_problems\_pmd\_diff\_output.csv

export countNewProblems=$(cat $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/pmd\_result\_$(Build.BuildId)/list\_new\_problems\_pmd\_diff\_output.csv | wc -l)

displayName: 'Get PMD Diff - Identify Problems'

# Publish PMD Analysis Results

- task: PublishPipelineArtifact@1

inputs:

targetPath: '$(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/pmd\_result\_$(Build.BuildId)'

artifactName: 'PMD\_Result\_$(System.JobAttempt)'

displayName: 'Publish PMD Analysis Results'

# List Plugins

- script: |

sfdx plugins --core

displayName: 'List Plugins'

- task: UseDotNet@2

displayName: 'Use .NET Core sdk'

inputs:

packageType: sdk

version: 3.1.x

installationPath: $(Agent.ToolsDirectory)/dotnet

- bash: |

cd $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)

echo -e "\n\n"

echo -e "--------------------------------------------------------------"

echo -e "==> package.xml generated with 'add' and 'modify' metadata"

export countComponentDeltaPackage=$(cat $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)/package/package.xml | egrep -i '<members>' | wc -l)

if [ $countComponentDeltaPackage -ge 1 ]; then

echo "##vso[task.setvariable variable=DELTA\_DEPLOY\_STATUS;isOutput=true]true"

else

echo "##vso[task.setvariable variable=DELTA\_DEPLOY\_STATUS;isOutput=true]false"

fi

echo "==> Count of Component in 'Delta ADD/MODIFY Package': $countComponentDeltaPackage"

export countNoTestDeltaPackage=$(cat $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)/package/package.xml | egrep -ic '<name>ApexClass</name>|<name>ApexTrigger</name>')

if [ $countNoTestDeltaPackage -ge 1 ]; then

echo "##vso[task.setvariable variable=NO\_TEST\_DELTA\_DEPLOY;isOutput=true]false"

echo "==> NO\_TEST\_DELTA\_DEPLOY: false"

else

echo "##vso[task.setvariable variable=NO\_TEST\_DELTA\_DEPLOY;isOutput=true]true"

echo "==> NO\_TEST\_DELTA\_DEPLOY: true"

fi

echo -e "--------------------------------------------------------------"

cat ./artifact\_$(Build.BuildId)/package/package.xml

echo -e "\n\n"

echo -e "--------------------------------------------------------------"

echo -e "==> destructiveChanges.xml generated with 'delete' metadata"

export countComponentDeltaDestructive=$(cat $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)/destructiveChanges/destructiveChanges.xml | egrep -i '<members>' | wc -l)

if [ $countComponentDeltaDestructive -ge 1 ]; then

echo "##vso[task.setvariable variable=DELTA\_DESTRUCTIVE\_DEPLOY\_STATUS;isOutput=true]true"

else

echo "##vso[task.setvariable variable=DELTA\_DESTRUCTIVE\_DEPLOY\_STATUS;isOutput=true]false"

fi

echo -e "==> Count of Component in 'Delta DESTRUCTIVE Package': $countComponentDeltaDestructive"

export countNoTestDeltaDestructive=$(cat $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)/destructiveChanges/destructiveChanges.xml | egrep -ic '<name>ApexClass</name>|<name>ApexTrigger</name>')

if [ $countNoTestDeltaDestructive -ge 1 ]; then

echo "##vso[task.setvariable variable=NO\_TEST\_DELTA\_DESTRUCTIVE\_DEPLOY;isOutput=true]false"

echo "==> NO\_TEST\_DELTA\_DESTRUCTIVE\_DEPLOY: false"

else

echo "##vso[task.setvariable variable=NO\_TEST\_DELTA\_DESTRUCTIVE\_DEPLOY;isOutput=true]true"

echo "==> NO\_TEST\_DELTA\_DESTRUCTIVE\_DEPLOY: true"

fi

echo -e "--------------------------------------------------------------\n"

cat ./artifact\_$(Build.BuildId)/destructiveChanges/destructiveChanges.xml

echo -e "\n\n"

echo -e "--------------------------------------------------------------\n"

echo "List contents in artifacts"

echo -e "--------------------------------------------------------------\n"

ls -R ./artifact\_$(Build.BuildId)/\*

displayName: '==> LIST CHANGES'

name: details\_in\_delta\_step

- bash: |

echo "DELTA\_DEPLOY\_STATUS: $(details\_in\_delta\_step.DELTA\_DEPLOY\_STATUS)"

echo "DELTA\_DESTRUCTIVE\_DEPLOY\_STATUS: $(details\_in\_delta\_step.DELTA\_DESTRUCTIVE\_DEPLOY\_STATUS)"

if [[ ( "$(details\_in\_delta\_step.DELTA\_DEPLOY\_STATUS)" == 'false' ) && ( "$(details\_in\_delta\_step.DELTA\_DESTRUCTIVE\_DEPLOY\_STATUS)" == 'false' ) ]]; then

echo "\*\*\* NO CHANGES DETECTED to VALIDATE, BUILD & DEPLOY... failing the pipeline \*\*\*"

exit 1

fi

displayName: 'Detect: changes?'

- bash: |

echo "Check for On-going Salesforce DEPLOY"

readarray execSOQLQuery < <(sfdx force:data:soql:query -q "SELECT Id, Status, StartDate, CompletedDate FROM DeployRequest where status = 'InProgress'" -t -u $(salesforceUsername))

echo "${execSOQLQuery[\*]}"

displayName: 'Check for On-going Salesforce DEPLOY'

# Validate Deploy Delta Package Change - Add/Modify/Delete

- bash: |

set -e

echo $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)

cd $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)

export SOURCE\_API\_VERSION=$(cat sfdx-project.json | jq -r .sourceApiVersion)

echo "In sfdx-project.json sourceApiVersion=$SOURCE\_API\_VERSION"

export PACKAGE\_DIRECTORIES=$(cat sfdx-project.json | jq -r .packageDirectories[].path)

mkdir -p $PACKAGE\_DIRECTORIES

if [[ ( "$(details\_in\_delta\_step.DELTA\_DEPLOY\_STATUS)" == 'false' ) && ( "$(details\_in\_delta\_step.DELTA\_DESTRUCTIVE\_DEPLOY\_STATUS)" == 'true' ) ]]; then

echo "\*\*\* Only DESTRUCTIVE CHANGES DETECTED to VALIDATE, BUILD & DEPLOY... \*\*\*"

exit 0

elif [[ ( "$(details\_in\_delta\_step.DELTA\_DEPLOY\_STATUS)" == 'true' ) && ( "$(details\_in\_delta\_step.DELTA\_DESTRUCTIVE\_DEPLOY\_STATUS)" == 'false' ) ]]; then

echo "\*\*\* Only ADD/MODIFY CHANGES DETECTED to VALIDATE, BUILD & DEPLOY... \*\*\*"

if [[ ("$(TESTLEVEL)" == 'RunSpecifiedTests') && !( -z "$(RUNTESTS)") ]]; then

echo "TESTLEVEL: $(TESTLEVEL)"

echo "RUNTESTS: $(RUNTESTS)"

readarray execDeltaDeployTest < <(sfdx force:source:deploy --apiversion $SOURCE\_API\_VERSION -l $(TESTLEVEL) --runtests $(RUNTESTS) --checkonly -x $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)/package/package.xml -u $(salesforceUsername) --verbose --wait 360)

elif [[ "$(TESTLEVEL)" =~ ^(NoTestRun|RunLocalTests)$ ]]; then

echo "TESTLEVEL: $(TESTLEVEL)"

readarray execDeltaDeployTest < <(sfdx force:source:deploy --apiversion $SOURCE\_API\_VERSION -l $(TESTLEVEL) --checkonly -x $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)/package/package.xml -u $(salesforceUsername) --verbose --wait 360)

fi

elif [[ ( "$(details\_in\_delta\_step.DELTA\_DEPLOY\_STATUS)" == 'true' ) && ( "$(details\_in\_delta\_step.DELTA\_DESTRUCTIVE\_DEPLOY\_STATUS)" == 'true' ) ]]; then

echo "\*\*\* BOTH ADD/MODIFY & DESTRUCTIVE CHANGES DETECTED to VALIDATE, BUILD & DEPLOY... \*\*\*"

cp $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)/destructiveChanges/destructiveChanges.xml $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)/package/destructiveChangesPost.xml

if [[ ("$(TESTLEVEL)" == 'RunSpecifiedTests') && !( -z "$(RUNTESTS)") ]]; then

echo "TESTLEVEL: $(TESTLEVEL)"

echo "RUNTESTS: $(RUNTESTS)"

readarray execDeltaDeployTest < <(sfdx force:source:deploy --apiversion $SOURCE\_API\_VERSION -l $(TESTLEVEL) --runtests $(RUNTESTS) --checkonly -x $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)/package/package.xml --postdestructivechanges $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)/package/destructiveChangesPost.xml -u $(salesforceUsername) --verbose --wait 360)

elif [[ "$(TESTLEVEL)" =~ ^(NoTestRun|RunLocalTests)$ ]]; then

echo "TESTLEVEL: $(TESTLEVEL)"

readarray execDeltaDeployTest < <(sfdx force:source:deploy --apiversion $SOURCE\_API\_VERSION -l $(TESTLEVEL) --checkonly -x $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)/package/package.xml --postdestructivechanges $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)/package/destructiveChangesPost.xml -u $(salesforceUsername) --verbose --wait 360)

fi

fi

echo "============= Output: Validation Exec log ===================================================="

echo "${execDeltaDeployTest[\*]}"

export checkDeployID=$(echo "${execDeltaDeployTest[\*]}" | egrep -i 'Deploy ID:'| wc -l)

export errorCount=$(echo "${execDeltaDeployTest[\*]}" | egrep -i 'Component Failures' | cut -d"]" -f1 | cut -d"[" -f2)

if [[ ( -z "$errorCount" || "$errorCount" -eq 0 ) && ( "$checkDeployID" -ne 0 ) ]]; then

echo "${execDeltaDeployTest[\*]}" | egrep -i 'Deploy ID:'

echo "TESTLEVEL: $(TESTLEVEL)"

else

echo "\*\*\* Failed Validation\*\*\*"

echo "DeployID status below:"

echo "${execDeltaDeployTest[\*]}" | egrep -i 'Deploy ID:'

echo "errorCount:$errorCount"

echo "TESTLEVEL: $(TESTLEVEL)"

exit 1

fi

echo "=============================================================================================="

for i in "${execDeltaDeployTest[@]}"

do

if [[ "$i" == "Deploy ID:"\* ]] ; then

export V\_DELTA\_DEPLOY\_ID=$(echo "${i:11:30}" | xargs)

echo "Delta\_Deploy\_ID: $V\_DELTA\_DEPLOY\_ID"

echo "##vso[task.setvariable variable=DELTA\_DEPLOY\_ID;isOutput=true]$V\_DELTA\_DEPLOY\_ID"

readarray outputDeltaDeployTest < <(sfdx force:source:deploy:report --apiversion $SOURCE\_API\_VERSION -i $V\_DELTA\_DEPLOY\_ID -u $(salesforceUsername) --wait 360 --verbose)

echo "------------- Output: Validation Report log ----------------------------------------------------"

echo "${outputDeltaDeployTest[\*]}"

export errorCount=$(echo "${outputDeltaDeployTest[\*]}" | egrep -i 'Component Failures' | cut -d"]" -f1 | cut -d"[" -f2)

if [[ (-z "$errorCount" || "$errorCount" -eq 0) ]]; then

echo "\*\*\* Successful Component Validation \*\*\*"

echo "TESTLEVEL: $(TESTLEVEL)"

else

echo "\*\*\* Failed Component Validation\*\*\*"

echo "TESTLEVEL: $(TESTLEVEL)"

echo "errorCount: $errorCount"

exit 1

fi

export successTestCount=$(echo "${outputDeltaDeployTest[\*]}" | egrep -i 'Test Success' | cut -d"]" -f1 | cut -d"[" -f2)

export failedTestCount=$(echo "${outputDeltaDeployTest[\*]}" | egrep -i 'Test Failures' | cut -d"]" -f1 | cut -d"[" -f2)

echo "${outputDeltaDeployTest[\*]}"

if [[ (-z "$failedTestCount" || "$failedTestCount" -eq 0) && ( -z "$successTestCount" || "$successTestCount" -ne 0 ) ]]; then

echo "TESTLEVEL: $(TESTLEVEL)"

if [[ ! -z "$successTestCount" ]]; then

echo "\*\*\* Successful Test \*\*\*"

echo "successTestCount: $successTestCount"

fi

else

echo "TESTLEVEL: $(TESTLEVEL)"

if [[ ! -z "$failedTestCount" ]]; then

echo "\*\*\* Failed Test \*\*\*"

echo "failedTestCount: $failedTestCount"

fi

exit 1

fi

echo "------------------------------------------------------------------------------------------------"

echo "${outputDeltaDeployTest[\*]}"

for j in "${outputDeltaDeployTest[@]}"

do

if [[ "$j" == \*"Status: Failed"\* ]] ; then

echo $j

echo "FAILED validation task with TESTLEVEL: $(TESTLEVEL)"

exit 1

elif [[ "$j" == \*"Status: Canceled"\* ]] ; then

echo $j

echo "CANCELED validation task TESTLEVEL: $(TESTLEVEL)"

exit 1

elif [[ "$j" == \*"Status: Succeeded"\* ]] ; then

echo $j

echo "SUCCEEDED validation task TESTLEVEL: $(TESTLEVEL)"

fi

done

fi

done

displayName: '==> VALIDATE: DELTA'

name: validate\_delta\_step

condition: and(eq(variables['details\_in\_delta\_step.NO\_TEST\_DELTA\_DEPLOY'], 'false' ), eq(variables['details\_in\_delta\_step.DELTA\_DEPLOY\_STATUS'], 'true' ))

# Validate Deploy Delta Package Change - Add/Modify/Delete

- bash: |

set -e

echo $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)

cd $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)

export SOURCE\_API\_VERSION=$(cat sfdx-project.json | jq -r .sourceApiVersion)

echo "In sfdx-project.json sourceApiVersion=$SOURCE\_API\_VERSION"

export PACKAGE\_DIRECTORIES=$(cat sfdx-project.json | jq -r .packageDirectories[].path)

mkdir -p $PACKAGE\_DIRECTORIES

if [[ ( "$(details\_in\_delta\_step.DELTA\_DEPLOY\_STATUS)" == 'false' ) && ( "$(details\_in\_delta\_step.DELTA\_DESTRUCTIVE\_DEPLOY\_STATUS)" == 'true' ) ]]; then

echo "\*\*\* Only DESTRUCTIVE CHANGES DETECTED to VALIDATE, BUILD & DEPLOY... \*\*\*"

exit 0

elif [[ ( "$(details\_in\_delta\_step.DELTA\_DEPLOY\_STATUS)" == 'true' ) && ( "$(details\_in\_delta\_step.DELTA\_DESTRUCTIVE\_DEPLOY\_STATUS)" == 'false' ) ]]; then

echo "\*\*\* Only ADD/MODIFY CHANGES DETECTED to VALIDATE, BUILD & DEPLOY... \*\*\*"

readarray execDeltaDeployTest < <(sfdx force:source:deploy --apiversion $SOURCE\_API\_VERSION --checkonly -x $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)/package/package.xml -u $(salesforceUsername) --verbose --wait 360)

elif [[ ( "$(details\_in\_delta\_step.DELTA\_DEPLOY\_STATUS)" == 'true' ) && ( "$(details\_in\_delta\_step.DELTA\_DESTRUCTIVE\_DEPLOY\_STATUS)" == 'true' ) ]]; then

echo "\*\*\* BOTH ADD/MODIFY & DESTRUCTIVE CHANGES DETECTED to VALIDATE, BUILD & DEPLOY... \*\*\*"

cp $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)/destructiveChanges/destructiveChanges.xml $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)/package/destructiveChangesPost.xml

readarray execDeltaDeployTest < <(sfdx force:source:deploy --apiversion $SOURCE\_API\_VERSION --checkonly -x $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)/package/package.xml --postdestructivechanges $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)/package/destructiveChangesPost.xml -u $(salesforceUsername) --verbose --wait 360)

fi

echo "============= Output: Validation Exec log ===================================================="

echo "${execDeltaDeployTest[\*]}"

export checkDeployID=$(echo "${execDeltaDeployTest[\*]}" | egrep -i 'Deploy ID:'| wc -l)

export errorCount=$(echo "${execDeltaDeployTest[\*]}" | egrep -i 'Component Failures' | cut -d"]" -f1 | cut -d"[" -f2)

if [[ ( -z "$errorCount" || "$errorCount" -eq 0 ) && ( "$checkDeployID" -ne 0 ) ]]; then

echo "${execDeltaDeployTest[\*]}" | egrep -i 'Deploy ID:'

echo "TESTLEVEL: $(TESTLEVEL)"

else

echo "\*\*\* Failed Validation\*\*\*"

echo "DeployID status below:"

echo "${execDeltaDeployTest[\*]}" | egrep -i 'Deploy ID:'

echo "errorCount:$errorCount"

echo "TESTLEVEL: $(TESTLEVEL)"

exit 1

fi

echo "=============================================================================================="

for i in "${execDeltaDeployTest[@]}"

do

if [[ "$i" == "Deploy ID:"\* ]] ; then

export V\_DELTA\_DEPLOY\_ID=$(echo "${i:11:30}" | xargs)

echo "Delta\_Deploy\_ID: $V\_DELTA\_DEPLOY\_ID"

echo "##vso[task.setvariable variable=DELTA\_DEPLOY\_ID;isOutput=true]$V\_DELTA\_DEPLOY\_ID"

readarray outputDeltaDeployTest < <(sfdx force:source:deploy:report --apiversion $SOURCE\_API\_VERSION -i $V\_DELTA\_DEPLOY\_ID -u $(salesforceUsername) --wait 360 --verbose)

echo "------------- Output: Validation Report log ----------------------------------------------------"

echo "${outputDeltaDeployTest[\*]}"

export errorCount=$(echo "${outputDeltaDeployTest[\*]}" | egrep -i 'Component Failures' | cut -d"]" -f1 | cut -d"[" -f2)

if [[ (-z "$errorCount" || "$errorCount" -eq 0) ]]; then

echo "\*\*\* Successful Component Validation \*\*\*"

echo "TESTLEVEL: $(TESTLEVEL)"

else

echo "\*\*\* Failed Component Validation\*\*\*"

echo "TESTLEVEL: $(TESTLEVEL)"

echo "errorCount: $errorCount"

exit 1

fi

export successTestCount=$(echo "${outputDeltaDeployTest[\*]}" | egrep -i 'Test Success' | cut -d"]" -f1 | cut -d"[" -f2)

export failedTestCount=$(echo "${outputDeltaDeployTest[\*]}" | egrep -i 'Test Failures' | cut -d"]" -f1 | cut -d"[" -f2)

echo "${outputDeltaDeployTest[\*]}"

if [[ (-z "$failedTestCount" || "$failedTestCount" -eq 0) && ( -z "$successTestCount" || "$successTestCount" -ne 0 ) ]]; then

echo "TESTLEVEL: $(TESTLEVEL)"

if [[ ! -z "$successTestCount" ]]; then

echo "\*\*\* Successful Test \*\*\*"

echo "successTestCount: $successTestCount"

fi

else

echo "TESTLEVEL: $(TESTLEVEL)"

if [[ ! -z "$failedTestCount" ]]; then

echo "\*\*\* Failed Test \*\*\*"

echo "failedTestCount: $failedTestCount"

fi

exit 1

fi

echo "------------------------------------------------------------------------------------------------"

echo "${outputDeltaDeployTest[\*]}"

for j in "${outputDeltaDeployTest[@]}"

do

if [[ "$j" == \*"Status: Failed"\* ]] ; then

echo $j

echo "FAILED validation task with TESTLEVEL: $(TESTLEVEL)"

exit 1

elif [[ "$j" == \*"Status: Canceled"\* ]] ; then

echo $j

echo "CANCELED validation task TESTLEVEL: $(TESTLEVEL)"

exit 1

elif [[ "$j" == \*"Status: Succeeded"\* ]] ; then

echo $j

echo "SUCCEEDED validation task TESTLEVEL: $(TESTLEVEL)"

fi

done

fi

done

displayName: '==> VALIDATE (No Tests Metadata): DELTA'

name: validate\_no\_test\_delta\_step

condition: and(eq(variables['details\_in\_delta\_step.NO\_TEST\_DELTA\_DEPLOY'], 'true' ), eq(variables['details\_in\_delta\_step.DELTA\_DEPLOY\_STATUS'], 'true' ))

# Validate Deploy - Delta Distructive Package Change - Delete

- bash: |

set -e

shopt -s nocasematch;

if [[ "$(details\_in\_delta\_step.DELTA\_DEPLOY\_STATUS)" == 'true' ]]; then

exit 0

fi

if [[ ( "$(details\_in\_delta\_step.DELTA\_DESTRUCTIVE\_DEPLOY\_STATUS)" == 'false' ) ]]; then

echo "\*\*\* NO CHANGES DETECTED in Destructive Delta Package, skipping Validate step... \*\*\*"

exit 0

fi

echo "\*\*\* Only DESTRUCTIVE CHANGES DETECTED to VALIDATE, BUILD & DEPLOY... \*\*\*"

echo "DELTA\_DESTRUCTIVE\_DEPLOY\_STATUS: $(details\_in\_delta\_step.DELTA\_DESTRUCTIVE\_DEPLOY\_STATUS)"

cd $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)

export SOURCE\_API\_VERSION=$(cat sfdx-project.json | jq -r .sourceApiVersion)

echo "In sfdx-project.json sourceApiVersion=$SOURCE\_API\_VERSION"

export PACKAGE\_DIRECTORIES=$(cat sfdx-project.json | jq -r .packageDirectories[].path)

mkdir -p $PACKAGE\_DIRECTORIES

if [[ ("$(TESTLEVEL)" == 'RunSpecifiedTests') && !( -z "$(RUNTESTS)") ]]; then

echo "TESTLEVEL: $(TESTLEVEL)"

echo "RUNTESTS: $(RUNTESTS)"

echo $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)

#cd $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)

readarray execDeltaDestructiveTest < <(sfdx force:mdapi:deploy --apiversion $SOURCE\_API\_VERSION -l $(TESTLEVEL) --runtests $(RUNTESTS) --checkonly -d $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)/destructiveChanges -u $(salesforceUsername) --verbose --wait 360)

elif [[ "$(TESTLEVEL)" =~ ^(NoTestRun|RunLocalTests)$ ]]; then

echo "TESTLEVEL: $(TESTLEVEL)"

echo $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)

#cd $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)

readarray execDeltaDestructiveTest < <(sfdx force:mdapi:deploy --apiversion $SOURCE\_API\_VERSION -l $(TESTLEVEL) --checkonly -d $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)/destructiveChanges -u $(salesforceUsername) --verbose --wait 360)

fi

echo "============= Output: Validation Exec log ===================================================="

echo "${execDeltaDestructiveTest[\*]}"

export checkJobID=$(echo "${execDeltaDestructiveTest[\*]}" | egrep -i 'Job ID'| wc -l)

export checkDeployID=$(echo "${execDeltaDestructiveTest[\*]}" | egrep -i 'Deploy ID:'| wc -l)

export errorCount=$(echo "${execDeltaDestructiveTest[\*]}" | egrep -i 'ERROR'| wc -l)

if [[ ( "$errorCount" -eq 0 ) && ( "$checkJobID" -ne 0 || "$checkDeployID" -ne 0 ) ]]; then

echo "${execDeltaDestructiveTest[\*]}" | egrep -i 'Job ID|Deploy ID'

echo "errorCount:$errorCount"

else

echo "\*\*\* Failed Validation\*\*\*"

echo "Job ID status below"

echo "${execDeltaDestructiveTest[\*]}" | egrep -i 'Job ID|Deploy ID'

echo "errorCount:$errorCount"

echo "TESTLEVEL: $(TESTLEVEL)"

exit 1

fi

echo "=============================================================================================="

for i in "${execDeltaDestructiveTest[@]}"

do

if [[ "$i" == "Job ID |"\* ]] ; then

export V\_DELTA\_DESTRUCTIVE\_DEPLOY\_ID=$(echo "${i:9:30}" | xargs)

echo "Delta\_Destructive\_Deploy\_ID: $V\_DELTA\_DESTRUCTIVE\_DEPLOY\_ID"

echo "##vso[task.setvariable variable=DELTA\_DESTRUCTIVE\_DEPLOY\_ID;isOutput=true]$V\_DELTA\_DESTRUCTIVE\_DEPLOY\_ID"

readarray outputDeltaDestructiveTest < <(sfdx force:mdapi:deploy:report --apiversion $SOURCE\_API\_VERSION -i $V\_DELTA\_DESTRUCTIVE\_DEPLOY\_ID -u $(salesforceUsername) --wait 360 --verbose)

echo "------------- Output: Validation Report log ---------------------------------------------"

echo "${outputDeltaDestructiveTest[\*]}"

export componentFailure=$(echo "${outputDeltaDestructiveTest[\*]}" | egrep -i 'Component Failures' | cut -d"]" -f1 | cut -d"[" -f2)

if [[ "$componentFailure" -ge 1 ]]; then

echo "\*\*\* Failed Validation: failed component=$componentFailure \*\*\*"

echo "TESTLEVEL: $(TESTLEVEL)"

exit 1

else

echo "\*\*\* Successful Validation: failed component=$componentFailure \*\*\*"

echo "TESTLEVEL: $(TESTLEVEL)"

fi

echo "----------------------------------------------------------------------------------------"

echo "${outputDeltaDestructiveTest[\*]}"

for j in "${outputDeltaDestructiveTest[@]}"

do

if [[ "$j" == \*"Status: Failed"\* ]] ; then

echo $j

echo "FAILED validation task with TESTLEVEL: $(TESTLEVEL)"

exit 1

elif [[ "$j" == \*"ERROR"\* ]] ; then

echo $j

echo "FAILED validation task TESTLEVEL: $(TESTLEVEL)"

exit 1

elif [[ "$j" == \*"Status: Canceled"\* ]] ; then

echo $j

echo "CANCELED validation task TESTLEVEL: $(TESTLEVEL)"

exit 1

elif [[ "$j" == \*"Status: Succeeded"\* ]] ; then

echo $j

echo "SUCCEEDED validation task TESTLEVEL: $(TESTLEVEL)"

fi

done

elif [[ "$i" == "Deploy ID:"\* ]] ; then

export V\_DELTA\_DESTRUCTIVE\_DEPLOY\_ID=$(echo "${i:11:30}" | xargs)

echo "Delta\_Destructive\_Deploy\_ID: $V\_DELTA\_DESTRUCTIVE\_DEPLOY\_ID"

echo "##vso[task.setvariable variable=DELTA\_DESTRUCTIVE\_DEPLOY\_ID;isOutput=true]$V\_DELTA\_DESTRUCTIVE\_DEPLOY\_ID"

readarray outputDeltaDestructiveTest < <(sfdx force:mdapi:deploy:report --apiversion $SOURCE\_API\_VERSION -i $V\_DELTA\_DESTRUCTIVE\_DEPLOY\_ID -u $(salesforceUsername) --wait 360 --verbose)

echo "------------- Output: Validation Report log ---------------------------------------------"

echo "${outputDeltaDestructiveTest[\*]}"

export componentFailure=$(echo "${outputDeltaDestructiveTest[\*]}" | egrep -i 'Component Failures' | cut -d"]" -f1 | cut -d"[" -f2)

if [[ "$componentFailure" -ge 1 ]]; then

echo "\*\*\* Failed Validation: failed component=$componentFailure \*\*\*"

echo "TESTLEVEL: $(TESTLEVEL)"

exit 1

else

echo "\*\*\* Successful Validation: failed component=$componentFailure \*\*\*"

echo "TESTLEVEL: $(TESTLEVEL)"

fi

echo "----------------------------------------------------------------------------------------"

echo "${outputDeltaDestructiveTest[\*]}"

for j in "${outputDeltaDestructiveTest[@]}"

do

if [[ "$j" == \*"Status: Failed"\* ]] ; then

echo $j

echo "FAILED validation task with TESTLEVEL: $(TESTLEVEL)"

exit 1

elif [[ "$j" == \*"ERROR"\* ]] ; then

echo $j

echo "FAILED validation task TESTLEVEL: $(TESTLEVEL)"

exit 1

elif [[ "$j" == \*"Status: Canceled"\* ]] ; then

echo $j

echo "CANCELED validation task TESTLEVEL: $(TESTLEVEL)"

exit 1

elif [[ "$j" == \*"Status: Succeeded"\* ]] ; then

echo $j

echo "SUCCEEDED validation task TESTLEVEL: $(TESTLEVEL)"

fi

done

elif [[ "$i" == \*"ERROR"\* ]] ; then

echo $i

echo "ERROR occured"

exit 1

fi

done

shopt -u nocasematch;

displayName: '==> VALIDATE: ONLY DESTRUCTIVE'

name: validate\_destructive\_delta\_step

condition: and(eq(variables['details\_in\_delta\_step.NO\_TEST\_DELTA\_DESTRUCTIVE\_DEPLOY'], 'false' ), eq(variables['details\_in\_delta\_step.DELTA\_DEPLOY\_STATUS'], 'false' ))

# Validate Deploy - Delta Distructive Package Change - Delete

- bash: |

set -e

shopt -s nocasematch;

if [[ "$(details\_in\_delta\_step.DELTA\_DEPLOY\_STATUS)" == 'true' ]]; then

exit 0

fi

if [[ ( "$(details\_in\_delta\_step.DELTA\_DESTRUCTIVE\_DEPLOY\_STATUS)" == 'false' ) ]]; then

echo "\*\*\* NO CHANGES DETECTED in Destructive Delta Package, skipping Validate step... \*\*\*"

exit 0

fi

echo "\*\*\* Only DESTRUCTIVE CHANGES DETECTED to VALIDATE, BUILD & DEPLOY... \*\*\*"

echo "DELTA\_DESTRUCTIVE\_DEPLOY\_STATUS: $(details\_in\_delta\_step.DELTA\_DESTRUCTIVE\_DEPLOY\_STATUS)"

cd $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)

export SOURCE\_API\_VERSION=$(cat sfdx-project.json | jq -r .sourceApiVersion)

echo "In sfdx-project.json sourceApiVersion=$SOURCE\_API\_VERSION"

export PACKAGE\_DIRECTORIES=$(cat sfdx-project.json | jq -r .packageDirectories[].path)

mkdir -p $PACKAGE\_DIRECTORIES

echo $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)

readarray execDeltaDestructiveTest < <(sfdx force:mdapi:deploy --apiversion $SOURCE\_API\_VERSION --checkonly -d $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId)/destructiveChanges -u $(salesforceUsername) --verbose --wait 360)

echo "============= Output: Validation Exec log ===================================================="

echo "${execDeltaDestructiveTest[\*]}"

export checkJobID=$(echo "${execDeltaDestructiveTest[\*]}" | egrep -i 'Job ID'| wc -l)

export checkDeployID=$(echo "${execDeltaDestructiveTest[\*]}" | egrep -i 'Deploy ID:'| wc -l)

export errorCount=$(echo "${execDeltaDestructiveTest[\*]}" | egrep -i 'ERROR'| wc -l)

if [[ ( "$errorCount" -eq 0 ) && ( "$checkJobID" -ne 0 || "$checkDeployID" -ne 0 ) ]]; then

echo "${execDeltaDestructiveTest[\*]}" | egrep -i 'Job ID|Deploy ID'

echo "errorCount:$errorCount"

else

echo "\*\*\* Failed Validation\*\*\*"

echo "Job ID status below"

echo "${execDeltaDestructiveTest[\*]}" | egrep -i 'Job ID|Deploy ID'

echo "errorCount:$errorCount"

echo "TESTLEVEL: $(TESTLEVEL)"

exit 1

fi

echo "=============================================================================================="

for i in "${execDeltaDestructiveTest[@]}"

do

if [[ "$i" == "Job ID |"\* ]] ; then

export V\_DELTA\_DESTRUCTIVE\_DEPLOY\_ID=$(echo "${i:9:30}" | xargs)

echo "Delta\_Destructive\_Deploy\_ID: $V\_DELTA\_DESTRUCTIVE\_DEPLOY\_ID"

echo "##vso[task.setvariable variable=DELTA\_DESTRUCTIVE\_DEPLOY\_ID;isOutput=true]$V\_DELTA\_DESTRUCTIVE\_DEPLOY\_ID"

readarray outputDeltaDestructiveTest < <(sfdx force:mdapi:deploy:report --apiversion $SOURCE\_API\_VERSION -i $V\_DELTA\_DESTRUCTIVE\_DEPLOY\_ID -u $(salesforceUsername) --wait 360 --verbose)

echo "------------- Output: Validation Report log ---------------------------------------------"

echo "${outputDeltaDestructiveTest[\*]}"

export componentFailure=$(echo "${outputDeltaDestructiveTest[\*]}" | egrep -i 'Component Failures' | cut -d"]" -f1 | cut -d"[" -f2)

if [[ "$componentFailure" -ge 1 ]]; then

echo "\*\*\* Failed Validation: failed component=$componentFailure \*\*\*"

echo "TESTLEVEL: $(TESTLEVEL)"

exit 1

else

echo "\*\*\* Successful Validation: failed component=$componentFailure \*\*\*"

echo "TESTLEVEL: $(TESTLEVEL)"

fi

echo "----------------------------------------------------------------------------------------"

echo "${outputDeltaDestructiveTest[\*]}"

for j in "${outputDeltaDestructiveTest[@]}"

do

if [[ "$j" == \*"Status: Failed"\* ]] ; then

echo $j

echo "FAILED validation task with TESTLEVEL: $(TESTLEVEL)"

exit 1

elif [[ "$j" == \*"ERROR"\* ]] ; then

echo $j

echo "FAILED validation task TESTLEVEL: $(TESTLEVEL)"

exit 1

elif [[ "$j" == \*"Status: Canceled"\* ]] ; then

echo $j

echo "CANCELED validation task TESTLEVEL: $(TESTLEVEL)"

exit 1

elif [[ "$j" == \*"Status: Succeeded"\* ]] ; then

echo $j

echo "SUCCEEDED validation task TESTLEVEL: $(TESTLEVEL)"

fi

done

elif [[ "$i" == "Deploy ID:"\* ]] ; then

export V\_DELTA\_DESTRUCTIVE\_DEPLOY\_ID=$(echo "${i:11:30}" | xargs)

echo "Delta\_Destructive\_Deploy\_ID: $V\_DELTA\_DESTRUCTIVE\_DEPLOY\_ID"

echo "##vso[task.setvariable variable=DELTA\_DESTRUCTIVE\_DEPLOY\_ID;isOutput=true]$V\_DELTA\_DESTRUCTIVE\_DEPLOY\_ID"

readarray outputDeltaDestructiveTest < <(sfdx force:mdapi:deploy:report --apiversion $SOURCE\_API\_VERSION -i $V\_DELTA\_DESTRUCTIVE\_DEPLOY\_ID -u $(salesforceUsername) --wait 360 --verbose)

echo "------------- Output: Validation Report log ---------------------------------------------"

echo "${outputDeltaDestructiveTest[\*]}"

export componentFailure=$(echo "${outputDeltaDestructiveTest[\*]}" | egrep -i 'Component Failures' | cut -d"]" -f1 | cut -d"[" -f2)

if [[ "$componentFailure" -ge 1 ]]; then

echo "\*\*\* Failed Validation: failed component=$componentFailure \*\*\*"

echo "TESTLEVEL: $(TESTLEVEL)"

exit 1

else

echo "\*\*\* Successful Validation: failed component=$componentFailure \*\*\*"

echo "TESTLEVEL: $(TESTLEVEL)"

fi

echo "----------------------------------------------------------------------------------------"

echo "${outputDeltaDestructiveTest[\*]}"

for j in "${outputDeltaDestructiveTest[@]}"

do

if [[ "$j" == \*"Status: Failed"\* ]] ; then

echo $j

echo "FAILED validation task with TESTLEVEL: $(TESTLEVEL)"

exit 1

elif [[ "$j" == \*"ERROR"\* ]] ; then

echo $j

echo "FAILED validation task TESTLEVEL: $(TESTLEVEL)"

exit 1

elif [[ "$j" == \*"Status: Canceled"\* ]] ; then

echo $j

echo "CANCELED validation task TESTLEVEL: $(TESTLEVEL)"

exit 1

elif [[ "$j" == \*"Status: Succeeded"\* ]] ; then

echo $j

echo "SUCCEEDED validation task TESTLEVEL: $(TESTLEVEL)"

fi

done

elif [[ "$i" == \*"ERROR"\* ]] ; then

echo $i

echo "ERROR occured"

exit 1

fi

done

shopt -u nocasematch;

displayName: '==> VALIDATE (No Tests Metadata): ONLY DESTRUCTIVE'

name: validate\_no\_test\_destructive\_delta\_step

condition: and(eq(variables['details\_in\_delta\_step.NO\_TEST\_DELTA\_DESTRUCTIVE\_DEPLOY'], 'true' ), eq(variables['details\_in\_delta\_step.DELTA\_DEPLOY\_STATUS'], 'false' ))

- bash: |

echo "Check for On-going Salesforce DEPLOY"

readarray execSOQLQuery < <(sfdx force:data:soql:query -q "SELECT Id, Status, StartDate, CompletedDate FROM DeployRequest where status = 'InProgress'" -t -u $(salesforceUsername))

echo "${execSOQLQuery[\*]}"

displayName: 'Check for On-going Salesforce DEPLOY'

- bash: |

cd $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)

sed -e 's/artifact\_$(Build.BuildId)//g' $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/pmd\_result\_$(Build.BuildId)/delta\_pmd\_output\_$(Build.BuildId).csv | awk -F "\"\*,\"\*" '{print $3}' | egrep -v 'File' | sort > $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/pmd\_result\_$(Build.BuildId)/delta\_pmd\_output.csv

sed -e 's/target\_sf\_org\_$(Build.BuildId)//g' $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/pmd\_result\_$(Build.BuildId)/target\_salesforce\_org\_branch\_pmd\_output\_$(Build.BuildId).csv | awk -F "\"\*,\"\*" '{print $3}' | egrep -v 'File' | sort > $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/pmd\_result\_$(Build.BuildId)/target\_salesforce\_org\_branch\_pmd\_output.csv

export countProblemsInDelta=$(cat $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/pmd\_result\_$(Build.BuildId)/delta\_pmd\_output.csv | wc -l)

echo "Count of Problems in Delta Pkg: $countProblemsInDelta"

echo "List of Problems in Delta Pkg:"

cat $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/pmd\_result\_$(Build.BuildId)/delta\_pmd\_output.csv | uniq -c | tee $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/pmd\_result\_$(Build.BuildId)/count\_component\_delta\_pmd\_output.txt

export countProblemsInTarget=$(cat $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/pmd\_result\_$(Build.BuildId)/target\_salesforce\_org\_branch\_pmd\_output.csv | wc -l)

echo "Count of Problems in Target SF Org branch: $countProblemsInTarget"

echo "List of Problems in Target:"

cat $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/pmd\_result\_$(Build.BuildId)/target\_salesforce\_org\_branch\_pmd\_output.csv | uniq -c | tee $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/pmd\_result\_$(Build.BuildId)/count\_component\_target\_salesforce\_org\_branch\_pmd\_output.txt

comm -1 -3 <(sort $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/pmd\_result\_$(Build.BuildId)/target\_salesforce\_org\_branch\_pmd\_output.csv) <(sort $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/pmd\_result\_$(Build.BuildId)/delta\_pmd\_output.csv) | tee $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/pmd\_result\_$(Build.BuildId)/list\_new\_problems\_pmd\_diff\_output.csv

export countNewProblems=$(cat $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/pmd\_result\_$(Build.BuildId)/list\_new\_problems\_pmd\_diff\_output.csv | wc -l)

if [[ "$countNewProblems" -ge 1 ]]; then

echo "New problems reported in Delta Pkg, which do not exit in Org branch, hence failing the Azure DevOps Pipeline. Please verify !"

echo "\*\*\* Count of NEW Problems PRESENT ONLY in the Delta Pkg: $countNewProblems \*\*\*"

echo "\*\*\* List of NEW Problems PRESENT ONLY in the Delta Pkg \*\*\*"

cat $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/pmd\_result\_$(Build.BuildId)/list\_new\_problems\_pmd\_diff\_output.csv

exit 1

else

echo "\*\*\* No NEW Problems Identified \*\*\*"

fi

displayName: '==> VERIFY PMD DIFF - Identify Problems'

- bash: |

cd $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)

rm -rf $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/target\_sf\_org\_$(Build.BuildId)

displayName: 'Cleanup local'

- task: SnykSecurityScan@1

displayName: Snyk Security Scan

condition: and(eq(variables['Build.Reason'], 'PullRequest'), notIn( '${{ parameters.environment\_name }}' , 'preprod' ), notIn( '${{ parameters.environment\_name }}' , 'prod' ))

continueOnError: true

inputs:

serviceConnectionEndpoint: 'snyk'

testType: 'app'

monitorWhen: 'always'

failOnIssues: '$(SNYK\_FAIL\_ON\_ISSUES)'

organization: '$(SNYK\_ORG)'

testDirectory: '$(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)'

additionalArguments: '-d --all-projects --detection-depth=30'

- bash: |

cd $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)

touch foundDeltafile.txt

if [ ! -f reference-list-profile-permissionsets.txt ]; then

echo "Not found reference-list-profile-permissionsets.txt in project root folder, failing 'PR Profile & Permission Checks'"

exit 1

else

echo "Found reference-list-profile-permissionsets.txt in project root folder, continue to check in delta files..."

fi

echo "----> Delta files with Profiles & Permissionsets"

readarray -t deltaFilesList <<< $(find $(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/artifact\_$(Build.BuildId) -type f |egrep 'default/profiles/|default/permissionsets/'| cut -d'.' -f1 | awk -F 'default/' '{print $2}')

for elementCheck in "${!deltaFilesList[@]}"; do

if [[ ${deltaFilesList[$elementCheck]} ]]; then

echo ${deltaFilesList[\*]}

else

echo "Build does not contain any profiles/permissionsets to compare"

exit 0

fi;

done

echo "Found profiles/permissionsets to compare...";

echo "----> Reference list of Profile & Permissionsets"

export textFile=$(cat reference-list-profile-permissionsets.txt | grep -v '#'|grep -v -e '^[[:space:]]\*$' | awk '{gsub(/^ +| +$/,"")} {print $0}')

readarray -t referenceList <<< "$textFile"

echo ${referenceList[\*]}

for x in "${deltaFilesList[@]}"; do

for y in "${referenceList[@]}"; do

if [[ "$x" == \*"$y"\* ]]; then

echo "===> $x is present in the delta files <==="

echo "$x" >> foundDeltafile.txt

echo "true" >> match.txt

fi

done

done

readarray -t foundDelta < <(cat foundDeltafile.txt)

strFoundDelta=$(IFS=,;echo "${foundDelta[\*]}")

echo "##vso[task.setvariable variable=found\_delta;isOutput=true]$strFoundDelta"

match\_out=$(cat match.txt | grep -v "^$" | grep 'true'| sort -u | wc -l)

if [ $match\_out -gt 0 ]; then

match\_found=true

echo "match\_found=$match\_found"

echo ${strFoundDelta[\*]}

else

match\_found=false

echo "match\_found=$match\_found"

fi

echo "##vso[task.setvariable variable=match\_found;isOutput=true]$match\_found"

displayName: '==> PR Check: Profiles & PermissionSets'

name: list\_match

condition: and(succeeded(), eq(variables['Build.Reason'], 'PullRequest'), eq(variables['Build.Repository.Name'], 'woolworthslimited/salesforce-hr'), or(and(eq('${{ parameters.target\_salesforce\_org\_branch}}', 'HRSIT'), eq('${{ parameters.environment\_name}}', 'hrsit')), and(eq('${{ parameters.target\_salesforce\_org\_branch}}', 'HRUAT'), eq('${{ parameters.environment\_name}}', 'hruat')), and(eq('${{ parameters.target\_salesforce\_org\_branch}}', 'preprod'), eq('${{ parameters.environment\_name}}', 'preprod'))))

- bash: |

DATA="{\"body\":\"Hi $NOTIFY\_GITHUB\_USER, $NOTIFY\_GITHUB\_TEAM, Changes to Profile/PermissionSets --> $(list\_match.found\_delta), please review this pull request $(System.PullRequest.PullRequestNumber)\"}"

curl -L \

-X POST \

-H "Accept: application/vnd.github+json" \

-H "Authorization: Bearer $(GITHUB\_TOKEN)" \

-H "X-GitHub-Api-Version: 2022-11-28" \

"https://api.github.com/repos/$(Build.Repository.Name)/issues/$(System.PullRequest.PullRequestNumber)/comments" \

-d "$DATA"

displayName: 'Add Reviewers to the PR'

name: add\_reviewer

condition: eq(variables['list\_match.match\_found'], 'true')

continueOnError: true

- task: Bash@3

displayName: ProfilePermissionset PR reviewer notification

condition: eq(variables['list\_match.match\_found'], 'true')

continueOnError: true

inputs:

filePath: './azure-devops-shared-automation/devops-scripts/utilities.sh'

workingDirectory: '$(Build.SourcesDirectory)/azure-devops-shared-automation/devops-scripts'

failOnStderr: true

arguments: google\_notification "$(googleChatChannelWebhookUrl\_PR)" "-------\n\*${{ parameters.environment\_name }} - Changes in Profile/PermissionSets\*\n $(list\_match.found\_delta)\n PR link https://github.com/$(Build.Repository.Name)/pull/$(System.PullRequest.PullRequestNumber)" "$(System.DefaultWorkingDirectory)/$(set\_vars.REPOSITORY\_NAME)/tmp1"

- task: Bash@3

displayName: Send Build Publish status notification

condition: succeededOrFailed()

continueOnError: true

inputs:

filePath: './azure-devops-shared-automation/devops-scripts/utilities.sh'

workingDirectory: '$(Build.SourcesDirectory)/azure-devops-shared-automation/devops-scripts'

failOnStderr: true

arguments: google\_notification "$(googleChatChannelWebhookUrl)" "-------\n\*$(Agent.JobStatus) ${{ parameters.environment\_name }} $BUILD\_REASON Build $(Build.BuildId)\*." "$(Build.SourcesDirectory)"