@Library(['adp-jenkins', 'aim-pipeline-library@aim\_qa\_library']) \_

import com.adp.aim.pipeline.EmailReportService

def emailService

def getTestCaseScenariosByType() {

def SINGLE\_API\_TEST\_LIST = "[ 'na-oidc-existent-connection', 'na-oidc-new-connection', 'na-oidc-adm-existent-connection', 'na-oidc-adm-new-connection', 'emea-oidc-existent-connection', 'emea-oidc-new-connection', 'na-saml-existent-connection', 'na-saml-new-connection', 'na-saml-adm-existent-connection', 'na-saml-adm-new-connection', 'emea-saml-existent-connection', 'emea-saml-new-connection' ]"

def SINGLE\_UI\_TEST\_LIST = "[ 'na-client-onboarding-ui', 'emea-client-onboarding-ui', 'apac-client-onboarding-ui', 'na-oidc-corp-new-federation-setup-ui', 'na-oidc-admin-new-federation-setup-ui', 'emea-oidc-corp-new-federation-setup-ui', 'emea-oidc-admin-new-federation-setup-ui', 'na-saml-corp-new-federation-setup-ui', 'na-saml-admin-new-federation-setup-ui', 'emea-saml-corp-new-federation-setup-ui', 'emea-saml-admin-new-federation-setup-ui', 'na-admin-new-user-provision-ui', 'na-admin-new-user-not-provisioned-deprovision-ui', 'na-admin-new-user-search-all-deprovision-ui', 'na-corp-existent-user-fifth-search-by-all-fields-deprovision-ui', 'na-corp-existent-users-first-fifth-deprovision-ui', 'na-corp-existent-users-pagination-deprovision-ui', 'na-corp-extend-users-pagination-deprovision-ui', 'na-corp-nonexistent-user-deprovision-ui', 'na-corp-existent-user-change-pid-by-name-ui', 'na-corp-existent-user-change-pid-by-employeeId-ui' ]"

def GROUP\_API\_TEST\_LIST = "[ 'all-existent-connections', 'all-new-connections', 'oidc-regression', 'saml-regression', 'mobile-federation' ]"

def GROUP\_UI\_TEST\_LIST = "[ 'all-tests-ui', 'all-client-onboarding-ui', 'all-federation-setup-ui', 'all-user-setup-ui' ]"

def UTILS\_TEST\_LIST = "[ 'clean-okta', 'validate-credentials' ]"

def PIPELINE\_UTILS\_TEST\_LIST = "[ 'validate-pipeline' ]"

return """

if (TYPE.equals('NO-HARM')) {

return [ 'all-connections' ]

} else if (TYPE.equals('REGRESSION')) {

return [ 'regression' ]

} else if (TYPE.equals('CO-MOBILE-FEDERATION')) {

return [ 'mobile-federation' ]

} else if (TYPE.equals('CO-L7')) {

return [ 'na-oidc-new-connection' ]

} else if (TYPE.equals('CO-SITEMINDER')) {

return [ 'na-saml-new-connection' ]

} else if (TYPE.equals('SINGLE-API')) {

return ${SINGLE\_API\_TEST\_LIST}

} else if (TYPE.equals('SINGLE-UI')) {

return ${SINGLE\_UI\_TEST\_LIST}

} else if (TYPE.equals('GROUP-API')) {

return ${GROUP\_API\_TEST\_LIST}

} else if (TYPE.equals('GROUP-UI')) {

return ${GROUP\_UI\_TEST\_LIST}

} else if (TYPE.equals('UTILS')) {

return ${UTILS\_TEST\_LIST}

} else if (TYPE.equals('PIPELINE')) {

return ${PIPELINE\_UTILS\_TEST\_LIST}

} else {

return [ 'Could not get tests' ]

}

"""

}

properties([

parameters([

choice(

name: 'Env',

description: 'choose respective ENV/DC for nexo-domain services, choose respective CG ENV for forward sync tests',

choices: ['FIT', 'DIT', 'DEV', 'IPE', 'IATDC1', 'IATDC2','UATDC2','PRODDC1','PRODDC2',

'CG\_NEXO\_DIT', 'CG\_NEXO\_FIT', 'CG\_NEXO\_IPE', 'CG\_NEXO\_IAT', 'CG\_NEXO\_UAT', 'CG\_NEXO\_PROD','ALL','OKTA']

),

booleanParam(name: 'Run\_NEXO\_APIs', defaultValue: false, description: 'Check to run Domain Services-NEXO APIs job'),

booleanParam(name: 'Run\_Test\_E2E', defaultValue: false, description: 'Run Watchmen Newman API Automation E2E Tests'),

choice(name: 'Test\_E2E', description: 'choose service to run tests', choices: ['oid-service', 'internal-application-service']),

booleanParam(name: 'Nexo\_No\_Harm\_Newman\_API\_Tests', defaultValue: false, description: 'Nexo\_No Harm\_Newman\_API\_Tests'),

[$class: 'ExtendedChoiceParameterDefinition',

name: 'Nexo\_No Harm\_Newman\_API\_Tests',

type: 'PT\_CHECKBOX',

multiSelectDelimiter: ',',

description: 'Choose one or more applications to test',

value: 'account-service,api-gateway,password-management-servicecim,ALL'

],

booleanParam(name: 'Run\_Watchmen\_UI\_Job', defaultValue: false, description: 'Check to run Watchmen UI Child Job'),

choice(name: 'test', choices: ['@spInitiated', '@crsso', '@all', '@crssoAuthorization', '@crssoSignInService', '@crssoMarketplace', '@universalSignin'], description: 'Select the test'),

choice(name: 'envToRun', choices: ['local', 'saucelabs'], description: 'Select the environment to run.'),

booleanParam(name: 'Run\_Risk\_Assessment\_Job', defaultValue: false, description: 'Run Risk Assessment Service Automation Child Job'),

choice(name: 'Version', choices: ['v1','v2']),

booleanParam(name: 'RUN\_FEDERATION\_TESTS', defaultValue: false, description: 'Run Federation Child Tests'),

choice(

name: 'TYPE',

description: 'Type of test to run',

choices: ['NO-HARM', 'REGRESSION', 'CO-MOBILE-FEDERATION', 'CO-L7', 'CO-SITEMINDER', 'SINGLE-API', 'SINGLE-UI', 'GROUP-API', 'GROUP-UI', 'UTILS', 'PIPELINE']

),

[$class: 'CascadeChoiceParameter',

choiceType: 'PT\_SINGLE\_SELECT',

description: 'Select test',

filterLength: 1,

filterable: true,

name: 'TEST',

referencedParameters: 'TYPE',

script: [

$class: 'GroovyScript',

fallbackScript: [

classpath: [],

sandbox: true,

script: "return [ 'Could not get tests' ]"

],

script: [

classpath: [],

sandbox: true,

script: getTestCaseScenariosByType()

]

]

],

string(name: 'Notification\_Email', defaultValue: 'stephan.raju.kombathula@adp.com', description: 'Email to notify on NEXO NO HARM job completion')

])

])

pipeline {

agent { label 'docker' }

stages {

stage('Initialize') {

steps {

script {

emailService = new EmailReportService(this)

}

}

}

stage('Parallel Jobs-1') {

parallel {

stage('Domain Services-NEXO APIs') {

when {

expression { params.Run\_NEXO\_APIs }

}

steps {

catchError(buildResult: 'UNSTABLE', stageResult: 'FAILURE') {

script {

def colors = ColorUtils()

printSeparator(this, 'Domain Services', 'START')

echo "${colors.SEPARATOR}------------------------ Starting Domain Services Job ------------------------${colors.RESET}"

echo "\u001B[34m[Domain Services-NEXO APIs] Starting downstream job...\u001B[0m"

echo "${colors.DOMAIN\_SERVICES}[Domain Services] Starting job execution...${colors.RESET}"

echo "${colors.DOMAIN\_SERVICES}[Domain Services] Environment: ${params.Env}${colors.RESET}"

def timeDelay = '2000'

if (!(params.Env in ['FIT', 'DIT', 'DEV', 'IPE', 'IATDC1', 'IATDC2','UATDC2','PRODDC1','PRODDC2'])) {

timeDelay = '2500'

}

echo "\u001B[36mJob: ${env.JOB\_NAME} | Stage: Domain Services-NEXO APIs | Running on agent: ${env.NODE\_NAME}\u001B[0m"

echo "\u001B[33mUsing time delay: ${timeDelay} ms for Env: ${params.Env}\u001B[0m"

def startTime = new Date().format("yyyy-MM-dd HH:mm:ss")

def stageStartTime = System.currentTimeMillis()

try {

def jaxBuild = build job: 'AIM\_QA/jenkins-NexoSmokeTest-docker', parameters: [

string(name: 'Env', value: params.Env),

string(name: 'Time', value: timeDelay)

], wait: true

echo "\u001B[32mSUCCESS: Downstream job completed successfully.\u001B[0m"

def duration = "${(System.currentTimeMillis() - stageStartTime) / 1000}s"

def DOMAIN\_RESULT = getBuildResult(jaxBuild)

env.DOMAIN\_RESULT=DOMAIN\_RESULT

env.DOMAIN\_URL = jaxBuild.absoluteUrl

env.DOMAIN\_DURATION = duration

env.DOMAIN\_START\_TIME = startTime

echo "Domain Services Build Result: ${env.DOMAIN\_RESULT}"

} catch (err) {

echo "\u001B[31mERROR: Downstream job failed! Reason: ${err}\u001B[0m"

def duration = "${(System.currentTimeMillis() - stageStartTime) / 1000}s"

env.DOMAIN\_RESULT = 'FAILURE'

env.DOMAIN\_URL = ''

// Capture detailed error information

emailService.captureJobError('DOMAIN', err.message, duration, startTime)

throw err

}

}

}

}

post {

always {

script {

def colors = ColorUtils()

printSeparator(this, 'Domain Services', 'COMPLETED')

echo "${colors.SEPARATOR}------------------------ Completed Domain Services Job ------------------------${colors.RESET}"

if (!params.Run\_NEXO\_APIs) {

emailext(

to: params.Notification\_Email,

subject: "Domain Services-NEXO APIs - SKIPPED",

body: "Domain Services-NEXO APIs was skipped as Run\_NEXO\_APIs is false.<br>Build: ${env.BUILD\_URL ?: (currentBuild?.absoluteUrl ?: 'N/A')}",

mimeType: 'text/html'

)

}

}

}

success {

script {

def colors = ColorUtils()

echo "\u001B[34m[Domain Services-NEXO APIs] Stage succeeded.\u001B[0m"

echo "${colors.DOMAIN\_SERVICES}[Domain Services] Stage succeeded${colors.RESET}"

emailext(

mimeType: 'text/html',

to: params.Notification\_Email,

subject: "Domain Services-NEXO APIs - SUCCESS - ${params.Env}",

body: """

<tr><td>Environment: ${params.Env}</td></tr>

<tr><td>Result: SUCCESS</td></tr>

<tr><td>Job: Domain Services-NEXO APIs</td></tr>

"""

)

}

}

failure {

script {

def colors = ColorUtils()

echo "\u001B[34m[Domain Services-NEXO APIs] Stage failed.\u001B[0m"

echo "${colors.DOMAIN\_SERVICES}[Domain Services] Stage failed${colors.RESET}"

emailext(

mimeType: 'text/html',

to: params.Notification\_Email,

subject: "Domain Services-NEXO APIs - FAILURE - ${params.Env}",

body: """

<tr><td>Environment: ${params.Env}</td></tr>

<tr><td>Result: FAILURE</td></tr>

<tr><td>Job: Domain Services-NEXO APIs</td></tr>

"""

)

}

}

}//post

}//domain

stage('Run Federation SSO Tests') {

when {

expression { params.RUN\_FEDERATION\_TESTS }

}

steps {

catchError(buildResult: 'UNSTABLE', stageResult: 'FAILURE') {

script {

printSeparator(this, 'Nexo No Harm-Federated SSO', 'START')

def colors = ColorUtils()

echo "${colors.FEDERATION}[Federation] Starting Federation SSO tests...${colors.RESET}"

echo "${colors.FEDERATION}[Federation] Test: ${params.TEST}${colors.RESET}"

echo "${colors.FEDERATION}[Federation] Type: ${params.TYPE}${colors.RESET}"

echo "${colors.FEDERATION}[Federation] Type: ${params.Env}${colors.RESET}"

// Trigger the dashboard E2E automation child job

def startTime = new Date().format("yyyy-MM-dd HH:mm:ss")

def stageStartTime = System.currentTimeMillis()

try{

def federationsso = build job: 'AIM/fedsso-dashboard-e2e-automation/feature%2FAIM-qa',

parameters: [

string(name: 'TEST', value: params.TEST),

string(name: 'ENV', value: params.Env),

string(name: 'EMAIL\_REPORT', value: params.Notification\_Email),

string(name: 'BRANCH\_NAME', value: 'feature/AIM-qa')

],

propagate: true,

wait: true

def duration = "${(System.currentTimeMillis() - stageStartTime) / 1000}s"

env.fedsso\_result = getBuildResult(federationsso)

env.fedsso\_absoluteUrl = federationsso.absoluteUrl

env.fedsso\_DURATION = duration

env.fedsso\_START\_TIME = startTime

echo "Federation SSO job result: ${env.fedsso\_result}"

echo "Federation SSO job URL: ${env.fedsso\_absoluteUrl}"

}

catch (err) {

def duration = "${(System.currentTimeMillis() - stageStartTime) / 1000}s"

env.fedsso\_result = 'FAILURE'

env.fedsso\_absoluteUrl = ''

// Capture detailed error information

emailService.captureJobError('FedSSO', err.message, duration, startTime)

throw err

}

}

}

}

post {

success {

emailext(

to: params.Notification\_Email,

subject: "FedSSO Job SUCCESS - ${params.TEST} - ${params.Env}",

body: "Federation SSO job completed successfully.<br>Test: ${params.TEST}<br>Env: ${params.Env}",

mimeType: 'text/html'

)

}

failure {

emailext(

to: params.Notification\_Email,

subject: "FedSSO Job FAILURE - ${params.TEST} - ${params.Env}",

body: "Federation SSO job failed.<br>Test: ${params.TEST}<br>Env: ${params.Env}",

mimeType: 'text/html'

)

}

}

}//Federation Close

stage('Platoon Nexo Newman Automation') {

when {

expression { params.Nexo\_No\_Harm\_Newman\_API\_Tests }

}

steps {

catchError(buildResult: 'UNSTABLE', stageResult: 'FAILURE') {

script {

printSeparator(this, 'Platoon Nexo Newman Automation', 'START')

def colors = ColorUtils()

echo "${colors.SEPARATOR}------------------------ Starting Newman Automation Job ------------------------${colors.RESET}"

echo "${colors.NEWMAN}[Platoon Newman] Starting Platoon Newman Automation...${colors.RESET}"

echo "${colors.NEWMAN}[Platoon Newman] Job: ${env.JOB\_NAME} | Running on agent: ${env.NODE\_NAME}${colors.RESET}"

def platoonEnvMap = [

'FIT' : 'na-fit',

'DIT' : 'na-dit',

'IATDC1' : 'na-dc1-iat',

'UATDC2' : 'na-dc2-uat',

'PRODDC1': 'na-dc1-prod'

]

def platoonEnv = platoonEnvMap[params.Env]

def allowedPlatoon = [

'na-dit': ['account-service', 'api-gateway', 'password-management-service', 'ALL'],

'na-fit': ['account-service', 'api-gateway', 'password-management-service', 'ALL'],

'na-dc1-iat': ['account-service', 'api-gateway', 'password-management-service', 'ALL'],

'na-dc2-uat': ['account-service', 'api-gateway', 'password-management-service', 'ALL'],

'na-dc1-prod': ['account-service', 'api-gateway', 'password-management-service', 'ALL'],

]

def envToRun = platoonEnv

def testsToRun = params["Nexo\_No Harm\_Newman\_API\_Tests"]?.tokenize(',') ?: []

def notAllowed = testsToRun.findAll { !(it in (allowedPlatoon[envToRun] ?: [])) }

if (notAllowed) {

error "Test(s) '${notAllowed.join(', ')}' not allowed for environment '${envToRun}'. Allowed: ${(allowedPlatoon[envToRun] ?: []).join(', ')}"

}

// --- END VALIDATION LOGIC ---

//def causes = currentBuild.getBuildCauses()

//def BUILD\_TRIGGER\_USER = causes ? causes[0]?.shortDescription : ""

def RUN\_TEST = true

//def BUILD\_TRIGGER\_BY = ""

// Store in env for use in post blocks and summary

env.NEWMAN\_ENV = envToRun

env.NEWMAN\_TEST = params["Nexo\_No Harm\_Newman\_API\_Tests"]

// env.NEWMAN\_TRIGGER = BUILD\_TRIGGER\_BY

echo "\u001B[33mNEWMAN ENV: ${env.NEWMAN\_ENV.toLowerCase()}\u001B[0m"

echo "\u001B[33mNEWMAN TEST: ${env.NEWMAN\_TEST}\u001B[0m"

echo "\u001B[33mNEWMAN TRIGGER: ${env.NEWMAN\_TRIGGER}\u001B[0m"

def startTime = new Date().format("yyyy-MM-dd HH:mm:ss")

def stageStartTime = System.currentTimeMillis()

try{

def newmanBuild = build job: 'AIM/aim-newman-e2e-automation/feature%2Faim\_qa', parameters: [

string(name: 'Env', value: env.NEWMAN\_ENV),

string(name: 'Test', value: env.NEWMAN\_TEST),

// string(name: 'BUILD\_TRIGGER\_BY', value: env.NEWMAN\_TRIGGER),

booleanParam(name: 'RunTest', value: RUN\_TEST)

],

wait: true

def duration = "${(System.currentTimeMillis() - stageStartTime) / 1000}s"

// Store results properly

env.NEWMAN\_RESULT = getBuildResult(newmanBuild)

env.NEWMAN\_URL = newmanBuild.absoluteUrl

env.NEWMAN\_DURATION = duration

env.NEWMAN\_START\_TIME = startTime

}

catch (err) {

def duration = "${(System.currentTimeMillis() - stageStartTime) / 1000}s"

env.NEWMAN\_RESULT = 'FAILURE'

env.NEWMAN\_URL = ''

// Capture detailed error information

emailService.captureJobError('NEWMAN\_API', err.message, duration, startTime)

throw err

}

}

}

}

post {

always {

script {

def colors = ColorUtils()

echo "${colors.SEPARATOR}------------------------ Completed Newman Automation Job ------------------------${colors.RESET}"

if (!params.Nexo\_No\_Harm\_Newman\_API\_Tests) {

emailext(

to: params.Notification\_Email,

subject: "Platoon Newman Automation - SKIPPED",

body: "Platoon Newman Automation was skipped as CG\_Newman\_APIs is false.\nBuild: ${env.BUILD\_URL ?: (currentBuild?.absoluteUrl ?: 'N/A')}",

mimeType: 'text/html'

)

}

}

}

success {

echo "\u001B[35m[Platoon Nexo Newman Automation] Stage succeeded.\u001B[0m"

emailext(

mimeType: 'text/html',

to: params.Notification\_Email,

subject: "Newman Automation Report - ${env.NEWMAN\_TEST} - ${(env.NEWMAN\_ENV ? env.NEWMAN\_ENV.toLowerCase() : 'unknown')} - SUCCESS",

body: """

<tr><td>Executed By: ${env.NEWMAN\_TRIGGER}</td></tr>

<tr><td>Test: ${env.NEWMAN\_TEST}</td></tr>

<tr><td>Environment: ${env.NEWMAN\_ENV}</td></tr>

"""

)

}

failure {

echo "\u001B[35m[Platoon Nexo Newman Automation] Stage failed.\u001B[0m"

emailext(

mimeType: 'text/html',

to: params.Notification\_Email,

subject: "Newman Automation Report - ${env.NEWMAN\_TEST} - ${(env.NEWMAN\_ENV ? env.NEWMAN\_ENV.toLowerCase() : 'unknown')} - FAILURE",

body: """

<tr><td>Executed By: ${env.NEWMAN\_TRIGGER}</td></tr>

<tr><td>Test: ${env.NEWMAN\_TEST}</td></tr>

<tr><td>Environment: ${env.NEWMAN\_ENV}</td></tr>

"""

)

}

}

} // closes newman

}//parallel

}//stage parallel-1

stage('Parallel Jobs-2') {

parallel {

stage('Trigger Watchmen Newman Automation E2E Tests') {

when {

expression { params.Run\_Test\_E2E }

}

steps {

catchError(buildResult: 'UNSTABLE', stageResult: 'FAILURE') {

script {

printSeparator(this, 'Watchmen Newman Automation E2E Tests', 'START')

def colors = ColorUtils()

echo "\u001B[38;5;93m[Watchmen Newman E2E] Starting Watchmen Newman API Automation E2E Tests...\u001B[0m"

echo "${colors.WATCHMEN\_E2E}[Watchmen E2E] Starting E2E tests...${colors.RESET}"

echo "${colors.WATCHMEN\_E2E}[Watchmen E2E] Test: ${params.Test\_E2E}${colors.RESET}"

try {

def testToRun = params.Test\_E2E

def childBuild = build(

job: 'AIM/watchmen-newman-e2e-automation/feature%2Faim\_qa',

parameters: [

string(name: 'Env', value: params.Env),

string(name: 'Test', value: testToRun),

string(name: 'emailRecipient', value: params.Notification\_Email)

],

propagate: false,

wait: true

)

env.CHILD\_RESULT = getBuildResult(childBuild)

env.CHILD\_URL = childBuild.absoluteUrl

env.CHILD\_ENV = params.Env

env.CHILD\_TEST = params.Test\_E2E

echo "Watchmen-newman-e2e-automation job result: ${env.CHILD\_RESULT}"

} catch (err) {

env.CHILD\_RESULT = 'FAILURE'

env.CHILD\_URL = ''

env.CHILD\_ENV = params.Env

env.CHILD\_TEST = params.Test\_E2E

error("Child job failed: ${err}")

}

}

} //catch error

} //steps

post {

always {

echo "\u001B[38;5;93m[Watchmen Newman E2E] Stage completed.\u001B[0m"

script {

def colors = ColorUtils()

echo "${colors.SEPARATOR}------------------------ Completed Watchmen Automation E2E Job ------------------------${colors.RESET}"

currentBuild.description = "${env.CHILD\_TEST} - ${env.CHILD\_ENV} (${env.CHILD\_RESULT})"

}

}

success {

echo "\u001B[38;5;93m[Watchmen Newman E2E] Stage succeeded.\u001B[0m"

emailext(

mimeType: 'text/html',

to: params.Notification\_Email,

subject: "PARENT: Watchmen Newman API Automation - ${env.CHILD\_TEST} - ${env.CHILD\_ENV} - ${env.CHILD\_RESULT}",

body: """

<b>Parent Pipeline completed successfully.</b><br>

<b>Child Job:</b> <a href="${env.CHILD\_URL}">${env.CHILD\_URL}</a><br>

<b>Test:</b> ${env.CHILD\_TEST}<br>

<b>Environment:</b> ${env.CHILD\_ENV}<br>

<b>Result:</b> ${env.CHILD\_RESULT}<br>

"""

)

}

failure {

echo "\u001B[38;5;93m[Watchmen Newman E2E] Stage failed.\u001B[0m"

emailext(

mimeType: 'text/html',

to: params.Notification\_Email,

subject: "PARENT: Watchmen Newman API Automation - ${env.CHILD\_TEST} - ${env.CHILD\_ENV} - ${env.CHILD\_RESULT}",

body: """

<b>Parent Pipeline FAILED.</b><br>

<b>Child Job:</b> <a href="${env.CHILD\_URL}">${env.CHILD\_URL}</a><br>

<b>Test:</b> ${env.CHILD\_TEST}<br>

<b>Environment:</b> ${env.CHILD\_ENV}<br>

<b>Result:</b> ${env.CHILD\_RESULT}<br>

"""

)

}

}

} // closes Newman Automation E2E

stage('Trigger Watchmen UI Job') {

when {

expression { params.Run\_Watchmen\_UI\_Job }

}

steps {

catchError(buildResult: 'UNSTABLE', stageResult: 'FAILURE') {

script {

def colors = ColorUtils()

echo "\u001B[38;5;93m[Watchmen Newman E2E] Starting Watchmen Newman UI Automation Tests...\u001B[0m"

echo "Job: ${env.JOB\_NAME} | Stage: Watchmen UI Automation Nexo | Running on agent: ${env.NODE\_NAME}"

echo "${colors.WATCHMEN\_UI}[Watchmen UI] Starting UI automation...${colors.RESET}"

echo "${colors.WATCHMEN\_UI}[Watchmen UI] Test: ${params.test}${colors.RESET}"

try{

def watchmenui = build job: 'NEXO/watchmen-ui-automation/feature%2Faim\_qa',

parameters: [

string(name: 'test', value: params.test),

string(name: 'environment', value: params.Env.toLowerCase()),

string(name: 'envToRun', value: params.envToRun),

string(name: 'emailReport', value: params.Notification\_Email)

],

propagate: true,

wait: true

env.UI\_CHILD\_RESULT = getBuildResult(watchmenui)

env.UI\_CHILD\_URL = watchmenui.absoluteUrl

env.UI\_CHILD\_TEST = params.test

env.UI\_CHILD\_ENV = params.environment

echo "\u001B[34m[Watchmen UI Parent] Child Result: ${env.UI\_CHILD\_RESULT}\u001B[0m"

echo "\u001B[34m[Watchmen UI Parent] Child URL: ${env.UI\_CHILD\_URL}\u001B[0m"

echo "\u001B[34m[Watchmen UI Parent] Child Test: ${env.UI\_CHILD\_TEST}\u001B[0m"

echo "\u001B[34m[Watchmen UI Parent] Child Environment: ${env.UI\_CHILD\_ENV}\u001B[0m"

}

catch (err) {

env.UI\_CHILD\_RESULT = 'FAILURE'

env.UI\_CHILD\_URL = ''

throw err

}

}

} //catch error

} //steps

post {

always {

echo "\u001B[34m[Watchmen UI Parent] Stage completed.\u001B[0m"

script {

def colors = ColorUtils()

echo "${colors.SEPARATOR}------------------------ Completed Watchmen Automation UI Job ------------------------${colors.RESET}"

currentBuild.description = "${env.UI\_CHILD\_TEST} - ${env.UI\_CHILD\_ENV} (${env.UI\_CHILD\_RESULT})"

}

}

success {

echo "\u001B[32m[Watchmen UI Parent] Stage succeeded.\u001B[0m"

emailext(

mimeType: 'text/html',

to: params.Notification\_Email,

subject: "PARENT: Watchmen UI Automation - ${env.UI\_CHILD\_TEST} - ${env.UI\_CHILD\_ENV} - ${env.UI\_CHILD\_RESULT}",

body: """

<b>Parent Pipeline completed successfully.</b><br>

<b>Child Job:</b> <a href="${env.UI\_CHILD\_URL}">${env.UI\_CHILD\_URL}</a><br>

<b>Test:</b> ${env.UI\_CHILD\_TEST}<br>

<b>Environment:</b> ${env.UI\_CHILD\_ENV}<br>

<b>Result:</b> ${env.UI\_CHILD\_RESULT}<br>

"""

)

}

failure {

echo "\u001B[31m[Watchmen UI Parent] Stage failed.\u001B[0m"

emailext(

mimeType: 'text/html',

to: params.Notification\_Email,

subject: "PARENT: Watchmen UI Automation - ${env.UI\_CHILD\_TEST} - ${env.UI\_CHILD\_ENV} - ${env.UI\_CHILD\_RESULT}",

body: """

<b>Parent Pipeline FAILED.</b><br>

<b>Child Job:</b> <a href="${env.UI\_CHILD\_URL}">${env.UI\_CHILD\_URL}</a><br>

<b>Test:</b> ${env.UI\_CHILD\_TEST}<br>

<b>Environment:</b> ${env.UI\_CHILD\_ENV}<br>

<b>Result:</b> ${env.UI\_CHILD\_RESULT}<br>

"""

)

}

}//watchmen ui post

}//watchmen ui

stage('Trigger Risk Assessment FSP Service Child Job') {

when {

expression { params.Run\_Risk\_Assessment\_Job }

}

steps {

catchError(buildResult: 'UNSTABLE', stageResult: 'FAILURE') {

script {

def colors = ColorUtils()

echo "${colors.SEPARATOR}------------------------ Started Risk Assessment FSP Service UI Job ------------------------${colors.RESET}"

echo "\u001B[34m[Risk Assessment Parent] Triggering child job: risk-assessment-service-automation\u001B[0m"

echo "${colors.RISK}[Risk Assessment] Starting risk assessment tests...${colors.RESET}"

echo "${colors.RISK}[Risk Assessment] Version: ${params.Version}${colors.RESET}"

def riskJob

try {

riskJob = build job: 'AIM/risk-assessment-service-automation/master',

parameters: [

string(name: 'Task', value: 'NoHarmTests'),

string(name: 'Version', value: params.Version),

string(name: 'Env', value: params.Env.toLowerCase()),

string(name: 'Branch', value: 'master')

],

propagate: true,

wait: true

env.RISK\_CHILD\_RESULT = getBuildResult(riskJob)

env.RISK\_CHILD\_URL = riskJob.absoluteUrl

echo "Risk Assessment FSP job result: ${env.RISK\_CHILD\_RESULT}"

} catch (err) {

env.RISK\_CHILD\_RESULT = 'FAILURE'

env.RISK\_CHILD\_URL = 'N/A'

echo "\u001B[31m[Risk Assessment Parent] Child job failed to start or complete: ${err}\u001B[0m"

}

env.RISK\_CHILD\_TASK = 'NoHarmTests'

env.RISK\_CHILD\_ENV = params.Env.toLowerCase()

echo "\u001B[34m[Risk Assessment Parent] Child Result: ${env.RISK\_CHILD\_RESULT}\u001B[0m"

echo "\u001B[34m[Risk Assessment Parent] Child URL: ${env.RISK\_CHILD\_URL}\u001B[0m"

echo "\u001B[34m[Risk Assessment Parent] Child Task: ${env.RISK\_CHILD\_TASK}\u001B[0m"

echo "\u001B[34m[Risk Assessment Parent] Child Environment: ${env.RISK\_CHILD\_ENV}\u001B[0m"

}

}

}

post {

always {

echo "\u001B[34m[Risk Assessment Parent] Stage completed.\u001B[0m"

script {

def colors = ColorUtils()

echo "${colors.SEPARATOR}------------------------ Completed Risk Assessment FSP Service UI Job ------------------------${colors.RESET}"

currentBuild.description = "${env.RISK\_CHILD\_TASK} - ${env.RISK\_CHILD\_ENV} (${env.RISK\_CHILD\_RESULT})"

}

}

success {

echo "\u001B[32m[Risk Assessment Parent] Stage succeeded.\u001B[0m"

emailext(

mimeType: 'text/html',

to: params.Notification\_Email,

subject: "PARENT: Risk Assessment Service Automation - ${env.RISK\_CHILD\_TASK} - ${env.RISK\_CHILD\_ENV} - ${env.RISK\_CHILD\_RESULT}",

body: """

<b>Parent Pipeline completed successfully.</b><br>

<b>Child Job:</b> <a href="${env.RISK\_CHILD\_URL}">${env.RISK\_CHILD\_URL}</a><br>

<b>Task:</b> ${env.RISK\_CHILD\_TASK}<br>

<b>Environment:</b> ${env.RISK\_CHILD\_ENV}<br>

<b>Result:</b> ${env.RISK\_CHILD\_RESULT}<br>

"""

)

}

failure {

echo "\u001B[31m[Risk Assessment Parent] Stage failed.\u001B[0m"

emailext(

mimeType: 'text/html',

to: params.Notification\_Email,

subject: "PARENT: Risk Assessment Service Automation - ${env.RISK\_CHILD\_TASK} - ${env.RISK\_CHILD\_ENV} - ${env.RISK\_CHILD\_RESULT}",

body: """

<b>Parent Pipeline FAILED.</b><br>

<b>Child Job:</b> <a href="${env.RISK\_CHILD\_URL}">${env.RISK\_CHILD\_URL}</a><br>

<b>Task:</b> ${env.RISK\_CHILD\_TASK}<br>

<b>Environment:</b> ${env.RISK\_CHILD\_ENV}<br>

<b>Result:</b> ${env.RISK\_CHILD\_RESULT}<br>

"""

)

}

}

}//fsp inception

}//parallel-2

}//stage parallel-2

}//stages

post {

always {

script {

try {

def total = getTotalJobs(this)

def success = getSuccessfulJobs(this)

def failed = getFailedJobs(this)

def skipped = getSkippedJobs(this)

echo "Total: ${total}, Success: ${success}, Failed: ${failed}, Skipped: ${skipped}"

def report = emailService.generateReport(params)

emailService.sendEmail(params, report)

echo "Successfully sent consolidated report"

} catch (e) {

echo "Failed to generate/send report: ${e.message}"

// Don't fail the build if report fails

}

}

}

}//consolidated post

}//pipeline