# CI/CD cho Verdaccio với Jenkins

Thiết lập hệ thống CI/CD cho Verdaccio. Tự động publish package khi có commit mới.

## Chuẩn bị

**Verdaccio Server**

**Jenkins**

- Jenkins Controller.
- Jenkins Agent.

**Jenkins Plugins**

- Blue Ocean
- NodeJS (Cài đặt tự động NodeJS lên Agent)
- Credentials
- Credentials Binding
- GitHub (Cần cho Webhook)
- Discord (Thông báo khi pipeline chạy xong)

**Package Repo** (Repo trên Git, Github, Gitlab... folder package chứa Jenkinsfile và package.json)

**Jenkins Credentials** (Cài đặt Credentials)

- Credential cho Verdaccio (chứa username, password đăng nhập Verdaccio)
- Credential cho Github (chứa username và PAT xác thực Github)
- Credential cho Github Secret (chứa Webhook Secret của Github)
- Credential cho Discord Webhook URL (chứa Webhook URL của Discord)

## Tạo và cấu hình Jenkins Jobs

Tạo item mới với type là **Pipeline**.

Cấu hình Jobs

- **General** (tuỳ chọn)
  - Description: mô tả
  - Github project: link đến github repo (không quan trọng)
- **Trigger**
  - Chọn **GitHub hook trigger for GITScm polling** để trigger từ GitHub.
  - **Poll SCM** để kiểm tra bào build định kỳ.
  - Để **trống** nếu muốn chạy thủ công.
- **Pipeline**
  - Definition:
    - **Pipeline script from SCM** để sử dụng jenkinsfile trong repo (recommend)
    - **Pipeline script** để viết trược tiếp jenkinsfile
  - SCM chọn Git
  - Repositories
    - Repository URL: URL của GitHub repo (Ex: `https://github.com/ThanhDang143/Backup.UniTask.git`)
    - Credentials: chọn GitHub Credentials đã tạo.
    - Branches to build: nhánh GitHub cần theo dõi
  - Script Path: Đường dẫn đến Jenkinsfile trong repo (mặc định là `Jenkinsfile`. Ex: `src/UniTask/Assets/Plugins/UniTask/Jenkinsfile`)

# Chạy và kiểm tra

Tại đây ta đã có thể chạy **build thủ công.**

Truy cập Blue Ocean. Để theo dõi tiến trình.

Kiểm tra kết quả và debug.

Nếu thành công thì có thể chuyển sang bước tiếp theo.

# Cấu hình Trigger tự động

Tiếp đến là cấu hình để Jenkins tự động chạy job khi có commit mới trên GitHub.

- Đảm bảo Jenkins Job đã chọn **GitHub hook trigger for GITScm polling** khi tạo Job.
- Cấu hình webhook trên GitHub Repo
  - Truy cập **GitHub Repo > Settings > Webhook > Add webhook**
  - **Payload URL**:
    - Mặc định là URL Jenkins theo sau là `/github-webhook/` (Ex: `https://jenkins.thanhdv.com/github-webhook/`)
    - URL Jenkins phải truy cập được từ internet.
  - **Content type**: chọn `application/json`
  - **Secret** là Credential cho Discord Webhook URL đã chuẩn bị

- **Which events**? chọn `Just the push event.`
- Đảm bảo đã tick chọn **Active** > **Add Webhook**

Sau khi add webhook github sẽ ping tới server. Kiểm tra tab **Recent Deliveries** để biết trạng thái ping

- Nếu không thành công kiểm tra lại các lỗi cơ bản như sai **Payload URL**, sai **Secret**, chưa **thêm secret vào Shared secrets**
- Nếu thành công có thể push một commit mới để test.

# Gửi thông báo qua Discord

Tạo Discord Webhoook, trên Discord truy cập **Server Settings > Integrations > Create Webhook.**

Copy Webhook URL đã tạo. Truy cập Jenkins Server tạo **Credential cho Discord Webhook URL**.

Trong bước tiếp theo ta cần cập nhật trong Jenkinsfile (*Đã có đầy đủ trong Jenkinsfile demo*)

- Tạo một biến môi trường `DISCORD_WEBHOOK = "CREDENTIAL_ID_DISCORD_WEBHOOK_URL"` để đảm bảo tính bảo mật không sử dụng trực tiếp Discord Webhook URL trong Jenkinsfile.
- Chỉnh sửa bên các khối sự kiện `success{ }, failure { }, aborted { }` bên trong khối `post` với nội dung demo như hình dưới. Chỉnh sửa lại nội dung cho phù hợp.

```
script {
  withCredentials([string(credentialsId: "${env.DISCORD_WEBHOOK}", variable:
'DISCORD_WEBHOOK_URL_SECRET')]) {
    discordSend(
      webhookURL: DISCORD_WEBHOOK_URL_SECRET,
      title: "?? Aborted: ${env.JOB_NAME}",
      description: "Build #${env.BUILD_NUMBER} for job `${env.JOB_NAME}` was aborted.",
      result: "ABORTED",
      link: env.BUILD_URL,
      footer: "Jenkins Build Notification"
    )
  }
}
```

# Phụ lục

## Cài ??t Credentials

### Thêm Domain (Tu? ch?n)

**Jenkins Manager > Credentials > System > Add domain**. Tại đây điền Name và Desc cho domain mới.

Nếu không tạo domain mới có thể sử dụng domain Global.

### Thêm Credentials

**Jenkins Manager > Credentials > System > Domain > Add Credentials**

- **Kind**
  - Username with password với Credential cho Verdaccio hoặc Credential cho Github...
  - Secret text với Credential cho Discord Webhook URL...
  - ...
- **Scope**: global
- **Username**
- **Password**: password hoặc PAT...
- **Secret**: chuỗi text
- **ID**: cần để sử dụng trong Jenkinsfile hoặc những nơi cần thiết...
- **Description**: mô tả.

**Chú ý:**

- Với **Credential cho GitHub Secret** sau khi tạo cần thêm vào **Shared secrets**
- **Jenkins Manager > System > GitHub > Advanced > Shared Secrets > Add Shared Secret** tại đây điền **ID** của Credential.
- Nếu không thấy **GitHub** kiểm tra lại GitHub Plugin

## L?i th??ng g?p khi ch?y Jenkins Job

Lỗi **`FileNotFoundException`**: Sai đường dẫn. Kiểm tra lại các path trong Jenkinsfile (PROJECT_PATH...)

Lỗi **`Cannot run program "sh"`**: sh là lệnh của Linux. Nếu máy Agent là Window thay `sh` bằng `bat`.

Lỗi **`RejectedAccessException`** (`new java.net.URI`): Bị chặn bởi Script Security sandbox. Phê duyệt tại **Jenkins Manager > In-process Script Approval**

## Jenkinsfile demo

### Windows

```
def packageVersionFromFile = ""
def packageName = ""
def verdaccioPackageUrl = ""

pipeline {
    // Agent with Node.js & Git required
    agent { label "thanhsserver" }

    // Ensure Node.js tool is available (configure in Jenkins Global Tool Config)
    tools {
        nodejs "NodeJS22"
    }

    environment {
        // Path to the directory containing package.json
        PROJECT_PATH = "src/UniTask/Assets/Plugins/UniTask"
```

```
        // Verdaccio registry URL
        VERDACCIO_REGISTRY_URL = "https://upm.thanhdv.com"
        // Jenkins Credential ID for Verdaccio auth (e.g., Secret Text or User/Pass with token)
        VERDACCIO_CREDENTIAL_ID = "THANHDV_VERDACCIO_AUTH"

        // Jenkins Credential ID for Git auth (MUST be "Username with password" type)
        // Username: your git username
        // Password: your git access token
        GIT_CREDENTIAL_ID = "THANHDV_GITHUB_JENKINS_CREDENTIAL"

        // Discord webhook create on Discord and add to Jenkins credential
        DISCORD_WEBHOOK = "DISCORD_VERDACCIO_WEBHOOK"

        // Release if commit has this key
        RELEASE_KEYWORD = "Release v"
    }
    // === End Configuration ===

    // Pipeline options
    options {
        timestamps()
        buildDiscarder(logRotator(numToKeepStr: "10"))
        timeout(time: 30, unit: "MINUTES")
        disableConcurrentBuilds()
    }

    stages {
        stage("Checkout") {
            steps {
                // Get source code
                checkout scm
                // git lfs pull // Uncomment if using Git LFS
            }
        }

        // stage("Validate Commit Message") {
        //     steps {
        //         dir(env.PROJECT_PATH) {
        //             script {
        //                 def isManualTrigger = false
        //                 currentBuild.getBuildCauses().each{ cause ->
        //                     if (cause instanceof hudson.model.Cause$UserIdCause ||
cause.shortDescription.contains("Started by user ")) {
        //                         isManualTrigger = true
        //                     }
        //                 }

        //                 if (isManualTrigger) {
        //                     echo "Build triggered by User! Ignore Validate Commit Message."
        //                 } else {
        //                     def commitMessage = bat(script: 'git log -1 --pretty=%%B', returnStdout:
true).trim()
        //                     echo "Checking commit message: ${commitMessage}"
        //                     if (!commitMessage.contains(RELEASE_KEYWORD)) {
        //                         echo "Build condition not met! Aborting pipeline..."
        //                         currentBuild.result = 'ABORTED'
        //                         return
        //                     }
        //                 }

        //                 echo "Commit message is valid for release."
        //             }
        //         }
        //     }
        // }
```

```
        stage("Prepare") {
            when {
                expression { return currentBuild.result != "ABORTED" }
            }

            steps {
                // Operate within the project directory
                dir(env.PROJECT_PATH) {
                    script {
                        // Read version directly from package.json
                        def pkg = readJSON file: "package.json" // Assumes package.json is at
PROJECT_PATH root
                        if (!pkg || !pkg.version || !pkg.name) {
                             error "Could not read version from package.json"
                        }

                        // Store the version in the script-level variable
                        packageVersionFromFile = pkg.version
                        echo "Package version: ${packageVersionFromFile}"

                        packageName = pkg.name
                        echo "Package name: ${packageName}"

                        verdaccioPackageUrl = "${env.VERDACCIO_REGISTRY_URL}/-/web/detail/${packageName}"
                        echo "Package URL: ${verdaccioPackageUrl}"

                        if (!packageVersionFromFile || !packageName || !verdaccioPackageUrl) {
                             error "Failed to set variable."
                        }

                        withCredentials([usernamePassword(credentialsId: env.VERDACCIO_CREDENTIAL_ID,
usernameVariable: "NPM_USER", passwordVariable: "NPM_PASS")]) {
                            echo "Configuring npm for Verdaccio using Username/Password..."
                            // Encrypt username:password to Base64
                            def userPass = "${NPM_USER}:${NPM_PASS}"
                            def encodedAuth =
java.util.Base64.getEncoder().encodeToString(userPass.getBytes("UTF-8"))

                            def registryUri = new URI(env.VERDACCIO_REGISTRY_URL)
                            def registryAuthority = registryUri.getAuthority()
                            def registryHostPath = "//${registryAuthority}/"

                            // Configure .npmrc for Verdaccio registry & auth token
                            bat "echo registry=${env.VERDACCIO_REGISTRY_URL} > .npmrc"
                            bat "echo ${registryHostPath}:_auth=\"${encodedAuth}\" >> .npmrc"
                            echo ".npmrc configured."
                        }

                        // Install dependencies (might run prepublish scripts)
                        // echo "Running npm install..."
                        // bat "npm install"
                    }
                }
            }
        }

        stage("Publish") {
            when {
                expression { return currentBuild.result != "ABORTED" }
            }

            steps {
                // Operate within the project directory
                dir(env.PROJECT_PATH) {
                    script {
                        // Use the version read from package.json
```

```
                        echo "Publishing package version ${packageVersionFromFile} to
${env.VERDACCIO_REGISTRY_URL}"
                        try {
                            // Publish using npm (reads package.json for name/version, uses .npmrc for
auth/registry)
                            bat "npm publish --registry ${env.VERDACCIO_REGISTRY_URL}"
                            echo "Package version ${packageVersionFromFile} published successfully!"
                        } catch (err) {
                            echo "ERROR: Failed to publish package!"
                            error "Publish failed: ${err.getMessage()}"
                        }
                    }
                }
            }
        }

        // // (Optional) Tag commit with the existing version from package.json
        // stage("Tag Existing Version") {
        //     // Only run on overall success so far
        //     when { expression { currentBuild.result == null || currentBuild.result == "SUCCESS" } }
        //     steps {
        //         // Operate within the project directory
        //         dir(env.PROJECT_PATH) {
        //             script {
        //                 echo "Tagging commit with existing version v${packageVersionFromFile}..."

        //                 // Inject Git credentials (Username = Git user, Password = Access Token)
        //                withCredentials([usernamePassword(credentialsId: env.GIT_CREDENTIAL_ID,
usernameVariable: "GIT_USERNAME", passwordVariable: "GIT_ACCESS_TOKEN")]) {

        //                     // Configure Git user (may not be needed if just tagging)
        //                     bat "git config user.email \"vanthanh1998@gmail.com\"" // EDIT Or use a
specific user
        //                     bat "git config user.name \"ThanhDVs Jenkins\""

        //                     // NO commit needed here as we didn"t change package.json version via npm
version

        //                     // Create annotated tag using the version from package.json
        //                     bat "git tag -a v${packageVersionFromFile} -m \"Release
v${packageVersionFromFile}\"" // Use the read version

        //                     // Push tag using HTTPS URL with embedded token
        //                     def repoUrl = scm.userRemoteConfigs[0].url
        //                     if (!repoUrl || !repoUrl.startsWith("https://")) {
        //                         error "Could not determine HTTPS repository URL from SCM
configuration."
        //                     }
        //                     def repoUrlClean = repoUrl.replaceAll(/https?:\/\/[^\/]+@/, "https://")
        //                     def pushUrl = repoUrlClean.replaceFirst("https://",
"https://${GIT_USERNAME}:${GIT_ACCESS_TOKEN}@")

        //                     // Push only the tag
        //                     bat "git push ${pushUrl}
refs/tags/v${packageVersionFromFile}:refs/tags/v${packageVersionFromFile}"

        //                     echo "Version tag v${packageVersionFromFile} pushed successfully."
        //                 }
        //             }
        //         }
        //     }
        // }
    }

    // Post-build actions
    post {
```

```
        // Always run cleanup
        always {
            echo "Build finished. Cleaning up..."
            // Clean up sensitive .npmrc file
            dir(env.PROJECT_PATH) {
                script {
                    try {
                        bat "del /F /Q .npmrc"
                    } catch (err) {
                        echo "Could not delete .npmrc (maybe it doesn't exist): ${err.getMessage()}"
                    }
                }
            }
        }

        // On success
        success {
            echo "Pipeline successful!"

            // Save artifacts if need
            // archiveArtifacts artifacts: '**/*.tgz', allowEmptyArchive: true

            echo "Cleaning up workspace..."
            deleteDir()

            echo "Sending success notification to Discord..."
            script {
                // Send
                withCredentials([string(credentialsId: "${env.DISCORD_WEBHOOK}", variable:
'DISCORD_WEBHOOK_URL_SECRET')]) {
                    discordSend(
                        webhookURL: DISCORD_WEBHOOK_URL_SECRET,
                        title: "? Success: ${env.JOB_NAME}",
                        description: "Job `${env.JOB_NAME}` build #${env.BUILD_NUMBER} published package
`${packageName}@${packageVersionFromFile}` successfully.\nBuild Log: ${env.BUILD_URL}.\nPackage URL:
${verdaccioPackageUrl}",
                        result: "SUCCESS",
                        link: env.BUILD_URL,
                        footer: "Jenkins Build Notification"
                    )
                }
            }
        }

        // On failure
        failure {
            echo "Pipeline failed!"

            echo "Sending failure notification to Discord..."
            script {
                // Send
                withCredentials([string(credentialsId: "${env.DISCORD_WEBHOOK}", variable:
'DISCORD_WEBHOOK_URL_SECRET')]) {
                    discordSend(
                        webhookURL: DISCORD_WEBHOOK_URL_SECRET,
                        title: "? Failure: ${env.JOB_NAME}",
                        description: "Job `${env.JOB_NAME}` build #${env.BUILD_NUMBER} failed to publish
package `${packageName}`.\nBuild Log: ${env.BUILD_URL}",
                        result: "FAILURE",
                        link: env.BUILD_URL,
                        footer: "Jenkins Build Notification"
                    )
                }
            }
        }
```

```
        aborted {
            echo "Pipeline aborted!"
            echo "Sending abort notification to Discord..."
            script {
                // Send
                withCredentials([string(credentialsId: "${env.DISCORD_WEBHOOK}", variable:
'DISCORD_WEBHOOK_URL_SECRET')]) {
                    discordSend(
                        webhookURL: DISCORD_WEBHOOK_URL_SECRET,
                        title: "?? Aborted: ${env.JOB_NAME}",
                        description: "Build #${env.BUILD_NUMBER} for job `${env.JOB_NAME}` was aborted.",
                        result: "ABORTED",
                        link: env.BUILD_URL,
                        footer: "Jenkins Build Notification"
                    )
                }
            }
        }

        unstable {
            echo "Pipeline unstable!"

            echo "Sending unstable notification to Discord..."
            script {
                withCredentials([string(credentialsId: "${env.DISCORD_WEBHOOK}", variable:
'DISCORD_WEBHOOK_URL_SECRET')]) {
                    discordSend(
                        webhookURL: DISCORD_WEBHOOK_URL_SECRET,
                        title: "?? Unstable: ${env.JOB_NAME}",
                        description: "Job `${env.JOB_NAME}` build #${env.BUILD_NUMBER} finished with
unstable status during processing of package `${packageName}`.\nBuild Log: ${env.BUILD_URL}",
                        result: "UNSTABLE",
                        link: env.BUILD_URL,
                        footer: "Jenkins Build Notification"
                    )
                }
            }
        }
    }
}
```

## Linux

```
def packageVersionFromFile = ""
def packageName = ""
def verdaccioPackageUrl = ""

pipeline {
    // Agent with Node.js & Git required
    agent { label "verdaccio-publisher" }

    // Ensure Node.js tool is available (configure in Jenkins Global Tool Config)
    tools {
        nodejs "NodeJS22"
    }

    environment {
        // Path to the directory containing package.json
        PROJECT_PATH = "Assets/Packages/DeviceDebugger"

        // Verdaccio registry URL
        VERDACCIO_REGISTRY_URL = "https://upm.thanhdv.com"
        // Jenkins Credential ID for Verdaccio auth (e.g., Secret Text or User/Pass with token)
        VERDACCIO_CREDENTIAL_ID = "THANHDV_VERDACCIO_AUTH"
```

```
        // Jenkins Credential ID for Git auth (MUST be "Username with password" type)
        // Username: your git username
        // Password: your git access token
        GIT_CREDENTIAL_ID = "THANHDV_GITHUB_JENKINS_CREDENTIAL"

        // Discord webhook create on Discord and add to Jenkins credential
        DISCORD_WEBHOOK = "DISCORD_VERDACCIO_WEBHOOK"

        // Release if commit has this key
        RELEASE_KEYWORD = "Release v"
    }
    // === End Configuration ===

    // Pipeline options
    options {
        timestamps()
        buildDiscarder(logRotator(numToKeepStr: "10"))
        timeout(time: 30, unit: "MINUTES")
        disableConcurrentBuilds()
    }

    stages {
        stage("Checkout") {
            steps {
                // Get source code
                checkout scm
                // git lfs pull // Uncomment if using Git LFS
            }
        }

        // stage("Validate Commit Message") {
        //     steps {
        //         dir(env.PROJECT_PATH) {
        //             script {
        //                 def isManualTrigger = false
        //                 currentBuild.getBuildCauses().each{ cause ->
        //                     if (cause instanceof hudson.model.Cause$UserIdCause ||
cause.shortDescription.contains("Started by user ")) {
        //                         isManualTrigger = true
        //                     }
        //                 }

        //                 if (isManualTrigger) {
        //                     echo "Build triggered by User! Ignore Validate Commit Message."
        //                 } else {
        //                     def commitMessage = sh(script: 'git log -1 --pretty=%%B', returnStdout:
true).trim()
        //                     echo "Checking commit message: ${commitMessage}"
        //                     if (!commitMessage.contains(RELEASE_KEYWORD)) {
        //                         echo "Build condition not met! Aborting pipeline..."
        //                         currentBuild.result = 'ABORTED'
        //                         return
        //                     }
        //                 }

        //                 echo "Commit message is valid for release."
        //             }
        //         }
        //     }
        // }

        stage("Prepare") {
            when {
                expression { return currentBuild.result != "ABORTED" }
            }
```

```
            steps {
                // Operate within the project directory
                dir(env.PROJECT_PATH) {
                    script {
                        // Read version directly from package.json
                        def pkg = readJSON file: "package.json" // Assumes package.json is at
PROJECT_PATH root
                        if (!pkg || !pkg.version || !pkg.name) {
                             error "Could not read version from package.json"
                        }

                        // Store the version in the script-level variable
                        packageVersionFromFile = pkg.version
                        echo "Package version: ${packageVersionFromFile}"

                        packageName = pkg.name
                        echo "Package name: ${packageName}"

                        verdaccioPackageUrl = "${env.VERDACCIO_REGISTRY_URL}/-/web/detail/${packageName}"
                        echo "Package URL: ${verdaccioPackageUrl}"

                        if (!packageVersionFromFile || !packageName || !verdaccioPackageUrl) {
                             error "Failed to set variable."
                        }

                        withCredentials([usernamePassword(credentialsId: env.VERDACCIO_CREDENTIAL_ID,
usernameVariable: "NPM_USER", passwordVariable: "NPM_PASS")]) {
                            echo "Configuring npm for Verdaccio using Username/Password..."
                            // Encrypt username:password to Base64
                            def userPass = "${NPM_USER}:${NPM_PASS}"
                            def encodedAuth =
java.util.Base64.getEncoder().encodeToString(userPass.getBytes("UTF-8"))

                            def registryUri = new URI(env.VERDACCIO_REGISTRY_URL)
                            def registryAuthority = registryUri.getAuthority()
                            def registryHostPath = "//${registryAuthority}/"

                            // Configure .npmrc for Verdaccio registry & auth token
                            sh "echo ${registryHostPath}:_auth=\"${encodedAuth}\" > .npmrc"
                            sh "echo registry=${env.VERDACCIO_REGISTRY_URL} >> .npmrc"
                            echo ".npmrc configured."
                        }

                        // Install dependencies (might run prepublish scripts)
                        echo "Running npm install..."
                        sh "npm install"
                    }
                }
            }
        }

        stage("Publish") {
            when {
                expression { return currentBuild.result != "ABORTED" }
            }

            steps {
                // Operate within the project directory
                dir(env.PROJECT_PATH) {
                    script {
                        // Use the version read from package.json
                        echo "Publishing package version ${packageVersionFromFile} to
${env.VERDACCIO_REGISTRY_URL}"
                        try {
                            // Publish using npm (reads package.json for name/version, uses .npmrc for
auth/registry)
```

```
                            sh "npm publish --registry ${env.VERDACCIO_REGISTRY_URL}"
                            echo "Package version ${packageVersionFromFile} published successfully!"
                        } catch (err) {
                            echo "ERROR: Failed to publish package!"
                            error "Publish failed: ${err.getMessage()}"
                        }
                    }
                }
            }
        }

        // // (Optional) Tag commit with the existing version from package.json
        // stage("Tag Existing Version") {
        //     // Only run on overall success so far
        //     when { expression { currentBuild.result == null || currentBuild.result == "SUCCESS" } }
        //     steps {
        //         // Operate within the project directory
        //         dir(env.PROJECT_PATH) {
        //             script {
        //                 echo "Tagging commit with existing version v${packageVersionFromFile}..."

        //                 // Inject Git credentials (Username = Git user, Password = Access Token)
        //                withCredentials([usernamePassword(credentialsId: env.GIT_CREDENTIAL_ID,
usernameVariable: "GIT_USERNAME", passwordVariable: "GIT_ACCESS_TOKEN")]) {

        //                     // Configure Git user (may not be needed if just tagging)
        //                     sh "git config user.email \"vanthanh1998@gmail.com\"" // EDIT Or use a
specific user
        //                     sh "git config user.name \"ThanhDVs Jenkins\""

        //                     // NO commit needed here as we didn"t change package.json version via npm
version

        //                     // Create annotated tag using the version from package.json
        //                     sh "git tag -a v${packageVersionFromFile} -m \"Release
v${packageVersionFromFile}\"" // Use the read version

        //                     // Push tag using HTTPS URL with embedded token
        //                     def repoUrl = scm.userRemoteConfigs[0].url
        //                     if (!repoUrl || !repoUrl.startsWith("https://")) {
        //                         error "Could not determine HTTPS repository URL from SCM
configuration."
        //                     }
        //                     def repoUrlClean = repoUrl.replaceAll(/https?:\/\/[^\/]+@/, "https://")
        //                     def pushUrl = repoUrlClean.replaceFirst("https://",
"https://${GIT_USERNAME}:${GIT_ACCESS_TOKEN}@")

        //                     // Push only the tag
        //                     sh "git push ${pushUrl}
refs/tags/v${packageVersionFromFile}:refs/tags/v${packageVersionFromFile}"

        //                     echo "Version tag v${packageVersionFromFile} pushed successfully."
        //                 }
        //             }
        //         }
        //     }
        // }
    }

    // Post-build actions
    post {
        // Always run cleanup
        always {
            echo "Build finished. Cleaning up..."
            // Clean up sensitive .npmrc file
            dir(env.PROJECT_PATH) {
```

```
                script {
                    try {
                        sh "rm -f .npmrc"
                    } catch (err) {
                        echo "Could not delete .npmrc (maybe it doesn't exist): ${err.getMessage()}"
                    }
                }
            }
        }

        // On success
        success {
            echo "Pipeline successful!"

            // Save artifacts if need
            // archiveArtifacts artifacts: '**/*.tgz', allowEmptyArchive: true

            echo "Cleaning up workspace..."
            deleteDir()

            echo "Sending success notification to Discord..."
            script {
                // Send
                withCredentials([string(credentialsId: "${env.DISCORD_WEBHOOK}", variable:
'DISCORD_WEBHOOK_URL_SECRET')]) {
                    discordSend(
                        webhookURL: DISCORD_WEBHOOK_URL_SECRET,
                        title: "? Success: ${env.JOB_NAME}",
                        description: "Job `${env.JOB_NAME}` build #${env.BUILD_NUMBER} published package
`${packageName}@${packageVersionFromFile}` successfully.\nBuild Log: ${env.BUILD_URL}.\nPackage URL:
${verdaccioPackageUrl}",
                        result: "SUCCESS",
                        link: env.BUILD_URL,
                        footer: "Jenkins Build Notification"
                    )
                }
            }
        }

        // On failure
        failure {
            echo "Pipeline failed!"

            echo "Sending failure notification to Discord..."
            script {
                // Send
                withCredentials([string(credentialsId: "${env.DISCORD_WEBHOOK}", variable:
'DISCORD_WEBHOOK_URL_SECRET')]) {
                    discordSend(
                        webhookURL: DISCORD_WEBHOOK_URL_SECRET,
                        title: "? Failure: ${env.JOB_NAME}",
                        description: "Job `${env.JOB_NAME}` build #${env.BUILD_NUMBER} failed to publish
package `${packageName}`.\nBuild Log: ${env.BUILD_URL}",
                        result: "FAILURE",
                        link: env.BUILD_URL,
                        footer: "Jenkins Build Notification"
                    )
                }
            }
        }

        aborted {
            echo "Pipeline aborted!"
            echo "Sending abort notification to Discord..."
            script {
                // Send
```

```
                withCredentials([string(credentialsId: "${env.DISCORD_WEBHOOK}", variable:
'DISCORD_WEBHOOK_URL_SECRET')]) {
                    discordSend(
                        webhookURL: DISCORD_WEBHOOK_URL_SECRET,
                        title: "?? Aborted: ${env.JOB_NAME}",
                        description: "Build #${env.BUILD_NUMBER} for job `${env.JOB_NAME}` was aborted.",
                        result: "ABORTED",
                        link: env.BUILD_URL,
                        footer: "Jenkins Build Notification"
                    )
                }
            }
        }

        unstable {
            echo "Pipeline unstable!"

            echo "Sending unstable notification to Discord..."
            script {
                withCredentials([string(credentialsId: "${env.DISCORD_WEBHOOK}", variable:
'DISCORD_WEBHOOK_URL_SECRET')]) {
                    discordSend(
                        webhookURL: DISCORD_WEBHOOK_URL_SECRET,
                        title: "?? Unstable: ${env.JOB_NAME}",
                        description: "Job `${env.JOB_NAME}` build #${env.BUILD_NUMBER} finished with
unstable status during processing of package `${packageName}`.\nBuild Log: ${env.BUILD_URL}",
                        result: "UNSTABLE",
                        link: env.BUILD_URL,
                        footer: "Jenkins Build Notification"
                    )
                }
            }
        }
    }
}
```

Revision #11
Created 2025-05-14 15:59:00 UTC by ThanhDV
Updated 2026-05-08 13:21:46 UTC by ThanhDV