**Steps For Integrating Gmail Notifications With Jenkins On Windows 10:**

**Prerequisites**

Before you start, you need to enable SMTP access in your Gmail settings to allow Jenkins to send emails using your Gmail account. Here’s how to do it:

1. Navigate to Manage Google account > Security.
2. Click on 2-Step Verification.
3. Scroll down to App Password and click on it.
4. Follow the instructions displayed to create an App password. Remember to save this password as it will be displayed only once.

For example, your App password might look something like this: inbz lqez dwfx dhxh.

**Plugin**

The Email Extension Plugin is included by default in recent versions of Jenkins, which simplifies the process of setting up email notifications for your builds.

**Credentials Setup**

To set up the credentials in Jenkins:

1. Go to Jenkins Dashboard > Manage Jenkins > Manage Credentials.
2. Click on (global) under Stores scoped to Jenkins.
3. Click on Add Credentials on the left side.
4. In the Kind dropdown, select Username with password.
5. In the Username field, enter your Gmail username. In the Password field, enter your App password (not your Gmail password).
6. In the ID field, enter a unique ID for this credential (e.g., gmail).
7. Click OK to save the credential.

**Email Notification Setup**

To configure the email notification:

1. Go to Manage Jenkins > Configure System.
2. Find the Email Notification tab and fill in the details:
3. SMTP server name: smtp.gmail.com
4. Default user e-mail suffix: @gmail.com
5. User name: user\_email\_id@gmail.com
6. Password: 123456
7. Use SSL: Checked
8. SMTP Port: 465

You can verify the email notification functionality by ticking the checkbox next to the Test configuration by sending Test e-mail recipient option. Enter a valid email id and click the Test configuration button to check whether the email id is valid or not.

**Extended E-mail Notification Setup**

To configure the extended email notification:

1. Go to Manage Jenkins > Configure System.
2. Find the Extended E-mail Notification tab and fill in the details:
3. SMTP server name: smtp.gmail.com
4. SMTP Port: 465
5. In the advanced tab, fill in the following details:
6. In the Credentials tab, upload/add the Gmail credentials that you created in the credential setup.
7. Use SSL: Checked
8. Default user e-mail suffix: @gmail.com

Jenkinsfile:  
pipeline {

agent any

stages {

stage('Hello') {

steps {

echo 'Hello World'

}

}

}

post {

always {

script{

env.BUILD\_TIMESTAMP = new Date(currentBuild.startTimeInMillis).format('MMMM dd, yyy | hh:mm:ss aaa | z')

}

emailext (

subject: "${currentBuild.currentResult} Build Report as of ${BUILD\_TIMESTAMP} — ${env.JOB\_NAME}",

body: """The Build report for ${env.JOB\_NAME} executed via Jenkins has finished its latest run.

- Job Name: ${env.JOB\_NAME}

- Job Status: ${currentBuild.currentResult}

- Job Number: ${env.BUILD\_NUMBER}

- Job URL: ${env.BUILD\_URL}

Please refer to the build information above for additional details.

This email is generated automatically by the system.

Thanks""",

recipientProviders: [[$class: 'DevelopersRecipientProvider']],

to: 'username@gmail.com',

attachLog: true

)

}

}

}

**Steps for GitHub webhook integration with Jenkins:**

**Prerequisites**

Before you start, ensure that you have the following:

1. An existing GitHub account and a project repository.
2. Jenkins installed and running on your system.

Note: The Git plugins come pre-installed with Jenkins, which simplifies the setup process.

**Configuration Steps**

Follow these steps to configure the GitHub webhook with Jenkins:

1. Copy the Jenkins URL. For example, http://localhost:9082. Note that in this example, a custom port is used instead of the default Jenkins port 8080.
2. Navigate to your project repository on GitHub and click on Settings at the repository level.
3. Go to Webhooks & services and click on Add webhook.
4. Fill in the required fields:

* Payload URL: Enter your Jenkins URL followed by /github-webhook. For example, <JenkinsURL>/github-webhook.
* Content type: Select application/json.

Leave the remaining fields as they are, unless there are specific requirements for your project.

**Steps for Make Localhost to public host using CloudFlare:**

**Prerequisites:**

Before you start, ensure that you have the following:

1. The CloudFlare application downloaded on your machine.
2. Jenkins installed and running on your system.

**How to use?**

1. Download cloudflared onto your machine. You can find the appropriate package for your operating system on the downloads page. For Windows, you can use this direct link to download the executable.

Direct link for download: - <https://github.com/cloudflare/cloudflared/releases/latest/download/cloudflared-windows-amd64.exe> or visit official page- <https://developers.cloudflare.com/cloudflare-one/connections/connect-networks/downloads/>

1. Once Downloaded package ,name- **cloudflared-windows-amd64.exe** & Rename the executable to **cloudflared.exe**
2. Open PowerShell, change the directory to your Downloads folder, and run the command **.\cloudflared.exe --version.** This should output the version of cloudflared, for example: **cloudflared version 2024.3.0 (built 2024-03-20-1009 UTC).**

Note: that cloudflared.exe could be cloudflared-windows-amd64.exe or cloudflared-windows-386.exe if you have not renamed it.

**Making Jenkins URL Public**

1. To make your Jenkins URL public:

Navigate to the CloudFlare directory and run the command **.\cloudflared.exe --url localhost:9081**. This will generate a public URL, for example, <https://sa-luxury-impacts-chevy.trycloudflare.com>.

**Why used Cloudflare** ?

Webhooks generally do not work with localhost:portnum as it only has local internet access. To use webhooks, you need to convert the localhost to a public host. After researching several applications, CloudFlare was found to be user-friendly and easy to set up.

**Note:**

You did not specify any valid additional argument to the cloudflared tunnel command.

If you are trying to run a Quick Tunnel then you need to explicitly pass the --url flag.

Eg. cloudflared tunnel --url localhost:8080/.

Please note that Quick Tunnels are meant to be ephemeral and should only be used for testing purposes.

For production usage, we recommend creating Named Tunnels. (https://developers.cloudflare.com/cloudflare-one/connections/connect-apps/install-and-setup/tunnel-guide/)

**Maven and Java configuration process:**

**Prerequisites:**

1. Before you start, you need to install Maven and Java packages. The installation process depends on your operating system. You can follow the instructions provided in the official documentation or online blogs:

Maven -- <https://phoenixnap.com/kb/install-maven-windows>

Java - <https://adoptium.net/temurin/releases/> or <https://phoenixnap.com/kb/install-java-windows>

Please note that the Maven and Java plugins are pre-installed with Jenkins, which simplifies the setup process.

**Configuration Steps in Jenkins Settings**

To configure Maven and Java in Jenkins, follow these steps:

* Go to Manage Jenkins > Global Tool Configuration.

**Maven Configuration**

Navigate to the Maven section.

1. Click on Maven installations... > Add Maven.
2. Fill in the required fields:
3. Name: Enter a name for this Maven installation (e.g., Maven).
4. MAVEN\_HOME: Enter the path where Maven is installed on your system (e.g., C:\Program Files\Maven-3.9.6\apache-maven-3.9.6).

**Java Configuration**

Navigate to the JDK section.

1. Click on JDK installations... > Add JDK.
2. Fill in the required fields:
3. Name: Enter a name for this JDK installation (e.g., Java22).
4. JAVA\_HOME: Enter the path where Java is installed on your system (e.g., C:\java\_home).

To verify the home locations of Maven and Java, you can use the **mvn -v** command in your system’s command line.

**The output will provide details about the installed versions of Maven and Java, including their home locations:**

Apache Maven 3.9.6 (bc0240f3c744dd6b6ec2920b3cd08dcc295161ae)

Maven home: **C:\Program Files\Maven-3.9.6\apache-maven-3.9.6**

Java version: 21.0.2, vendor: Eclipse Adoptium, runtime**: C:\java\_home**

Default locale: en\_IN, platform encoding: UTF-8

OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"

**The Steps to Integrate Docker with Jenkins on Windows 10:**

**Prerequisites:**

* Install Docker on your local system. In this context, Docker Desktop is installed on Windows 10.

**Plugins:**

1. Docker Pipeline: This plugin allows you to build and use Docker containers from pipelines.
2. Docker Plugin: This plugin integrates Jenkins with Docker.

**Install the Docker plugins on Jenkins:**

1. Log in to your Jenkins dashboard.
2. Navigate to Manage Jenkins > Manage Plugins.
3. Search for the Docker Pipeline and the Docker plugin and install them.

**Credentials Setup:**

1. Go to Jenkins Dashboard > Manage Jenkins > Manage Credentials.
2. Click on (global) under Stores scoped to Jenkins.
3. Click on Add Credentials on the left side.
4. In the Kind dropdown, select Username with password.
5. In the Username field, enter your Docker username and in the Password field, enter your Docker token (instead of password).
6. In the ID field, enter a unique ID for this credential (e.g., dockerpwd).
7. Click OK to save the credential.

After setting up everything, verify it using a pipeline and make sure both Jenkins and Docker are running.

**Jenkinsfile:**

pipeline {

agent any

stages {

stage('Hello') {

steps {

echo 'Hello World'

}

}

stage("verify docker"){

steps{

script {

withCredentials([usernamePassword(credentialsId: 'dockerpwd', usernameVariable: 'DOCKER\_USERNAME', passwordVariable: 'DOCKER\_PASSWORD')]) {

bat ' docker login --username %DOCKER\_USERNAME% --password %DOCKER\_PASSWORD% '

bat "docker version"

bat "docker ps "

bat "docker images"

}

}

}

}

}}

**Steps for Trivy install :**

1. Make sure Chocolate package manager installed or not in local system (windows 10)

If not, install it.

Link - <https://chocolatey.org/install> --

1. Use the command: choco install trivy – its downloaded! And verify it- **trivy version**

Usage: **trivy image <imagename>**

**The Steps to Integrate Kubernetes with Jenkins on Windows 10:**

**Prerequisites:**

1. Install Kubernetes on your local system. The installation process varies depending on the operating system. In this context, Kubernetes is installed using Docker Desktop.
2. Gather necessary information such as Kubernetes URL, config, and CA certificate. These files are typically located in the .kube directory.

**Plugins:**

1. Kubernetes CLI Plugin: This plugin is used to configure kubectl for Kubernetes.
2. Kubernetes Plugin: This plugin integrates Jenkins with Kubernetes.

**Install the Kubernetes plugins on Jenkins:**

1. Login to your Jenkins dashboard.
2. Navigate to Manage Jenkins > Manage Plugins.
3. Search for the Kubernetes plugin and the Kubernetes CLI plugin and install them.

**Credentials Setup:**

1. Create a secret file credential in Jenkins:
2. Go to Jenkins Dashboard > Manage Jenkins > Manage Credentials.
3. Click on (global) under Stores scoped to Jenkins.
4. Click on Add Credentials on the left side.
5. In the Kind dropdown, select Secret file.
6. In the Secret file, upload the config file (which is under the .kube directory).
7. In the ID field, enter a unique ID for this credential (e.g., my-credentials).
8. Click OK to save the credential.

After setting up everything, verify it using a pipeline and make sure both Jenkins and the Kubernetes application are running.

**Jenkinsfile:**   
pipeline {

agent any

stages {

stage('Hello') {

steps {

echo 'Hello World'

}

}

stage("verify k8s"){

steps{

withCredentials([file(credentialsId: 'config', variable: 'CA\_CERTIFICATE')]) {

kubeconfig(

credentialsId: 'config',

serverUrl: '',

caCertificate: '%env.CA\_CERTIFICATE%'

) {

bat 'kubectl get node'

// some block

}

}

}

}

}

}