**Methodology Documentation**

**Security Integration Impact Assessment Tool for Continuous Software Deployment Pipelines**

# A.1 Jenkins Summary

The following sections provide an overview of Jenkins.

## A.1.1 introduction

Jenkins is an automation program that allows users to build, test and deploy software using pipelines. This extends beyond code, meaning that Jenkins can be used for many other tasks including running bash and python scripts. Jenkins provides a web GUI and customisation that provides most of the functionality a user could need.

## A.1.2 Infrastructure

There is a master server which controls the pipelines and schedules builds to agents. Then, there are Agents/Nodes which perform the build in their respective workspace.

An example of the Jenkins infrastructure is outlined below:

* 1. Developer commits code to a git repository.
  2. The Jenkins master becomes aware of this command and triggers pipeline to one of the agents to run (Selects agent based on configured labels).
  3. Agent runs build (usually an array of Linux commands to build, test and distribute code).

Note that agents are operating system specific. This means that a pipeline created with Linux command will not be able to run with a Windows based agent and vice versa.

## A.1.3 Agent Types

There are two main agent types:

1. Permanent Agents – Dedicated servers for running jobs.
2. Cloud Agents – Dynamic Agents spun up on demand.

## A.1.4 Accessing Jenkins

For our project, Jenkins was setup on an AWS server and thus, some of the following descriptions are specific to this implementation.

Jenkins can be accessed by using the IP address of the AWS server instance and the port 8080. This will bring you to the login page where you need to provide your Jenkins credentials to login.

The AWS instance was subject to failing. The following section provides some common fixes.

## A.1.5 AWS Diagnotics

The following sections outline details on how to address AWS problems.

### A.1.5.1 AWS Instance Crash

An AWS instance crash occurs when a user is unable to connect to the instance. Furthermore, Jenkins will not load. The following steps can be performed to restart the instance.

* 1. Login to this link AWS EC2
  2. Select Instance State
  3. Select Reboot (If this doesn’t work you can try stop and force stop and then start, **DO NOT CLICK TERMINATE**), you may need to wait a few minutes for it to stop and a few minutes for it to start.
  4. Click connect.
  5. Type “sudo systemctl start jenkins” (you may need to try: sudo systemctl stop jenkins, and then start if it isn’t working still).
  6. Connect to the new IP address with the Jenkins port specified..

### A.1.5.2 More Difficult Issues

There are some instances where the fix is not that easy. This tutorial outlines how to reinstall Jenkins in such an instant. <https://www.jenkins.io/doc/tutorials/tutorial-for-installing-jenkins-on-AWS/>

# A.2 Basic Jenkins Pipeline

This section outlines the creation of a basic Jenkins pipeline in a procedural form.

1. Open Jenkins.
2. Select “New Item” on the homepage.
3. Type in your desired item name and “Pipeline” from the options below.
4. Select “OK”
5. Add your desired description (and build triggers if appropriate) and then scroll down to scripts and enter the following:

pipeline {  
 agent any   
   
 stages {  
 stage('Build') {  
 steps {  
 echo 'Building...'  
 }  
 }  
 stage('Test') {  
 steps {  
 echo 'Testing...'  
 }  
 }  
 stage('Deploy') {  
 steps {  
 echo 'Deploying...'  
 }  
 }  
 }  
}

1. Select "Save”
2. Select “Build Now” from the left-hand toolbar and wait around 3 seconds after which a table should then appear with four columns, three of which are titled “Build”, “Test” and “Deploy”.

To confirm each of the aforementioned options worked please repeat the following for each option:

* 1. Hover over the section (Most likely green) for the relevant option you wish to confirm is working, after which a pop up should appear.
  2. Select “Logs” from the aforementioned pop up menu.
  3. Another pop up should now appear on your screen indicating a print message that should be one of the following (depending on the option selected):
     1. “Building...”
     2. “Testing...”
     3. “Deploying...”
  4. Once all three have been checked this indicates that each stage was successful that that Jenkins is working.

# A.3 Setting Up An Agent

General disclaimer about nodes: Nodes generally will not work if the build agent is off because technically it is the master node and they are the slaves. However. this means you should write your pipeline such that it only uses agents with a label that works for your agent as the built-in node will probably crash for most pipelines.

The following sections are a description on how to create a node.

## A.3.1 Linux Based Node

1. From the dashboard select “Manage Jenkins”.
2. Then select “Nodes”.
3. Then select “New Node”.
4. Name it anything appropriate e.g Agent1.
5. Select “Permanent Agent”.
6. Select "Create”.
7. Add the address of the remote directory on your computer that you wish to use into the “Remote Root Directory”. Recommended in a location that makes the file easily accessible.
8. Fill in the “Labels” section with the same name as your agent’s name entered earlier.
9. Tick web socket.
10. Leave everything else as is already set as default and select “save”.
11. Then on the "Nodes” page click the name of the node you just created.
12. An error may appear saying something relating to a “JNLP agent port”, if so, click the blue hyper link to the right and follow the next two steps, if not then skip to step 14.
13. Under the “Agents” section of this page then set the TCP port from “Disabled” to “Fixed” with the port value 50000.
14. Select “Save” and then return to the “Nodes” page and click on the name of your recently created agent again.
15. Install java locally on your device with the following commands (for windows users, go to the Setting Up a Windows Based Agent section below:
    1. sudo rpm --import <https://yum.corretto.aws/corretto.key>
    2. sudo curl -L -o /etc/yum.repos.d/corretto.repo <https://yum.corretto.aws/corretto.repo>
    3. sudo yum install -y java-11-amazon-corretto-devel

If that doesn’t work, try: sudo mkdir -p /etc/yum.repos.d/ then repeat.

1. Copy the relevant command line commands as shown on the main page of the node you have just created and paste into your cloudshell (NOTE if the IP address in this command doesn’t match your instance IP address do the following.
   1. In your Jenkins Dashboard go to **Manage Jenkins > Configure System**, Under **Jenkins Location** set the **Jenkins URL** to the new IP address of your server computer, Save changes.
2. Refresh the main page for your node and it should be running.
3. To test select create a new free style project
4. Select a name for it.
5. Select “Restrict where this project can be run” and then from the search bar select your node.
6. Go to “Build Steps” and select execute shell and paste the following command in and select run:

uptime

echo $WORKSPACE

1. Refresh page to confirm success and additionally look at console to see a time appear as it should from uptime.

## A.3.2 Setting Up a Windows Based Agent

If you are using a windows-based agent, you must ensure that the Jenkins file uses the appropriate windows command such as ‘bat’ as most of the pipelines written so far have been for Linux.

1. From the dashboard select “Manage Jenkins”.
2. Then select “Nodes”.
3. Then select “New Node”.
4. Name it anything appropriate e.g Agent1.
5. Select “Permanent Agent”.
6. Select "Create”.
7. Add the address of the remote directory on your computer that you wish to use into the “Remote Root Directory”. Recommended in a location that makes the file easily accessible.
8. Fill in the “Labels” section with the same name as your agent’s name entered earlier.
9. Tick web socket.
10. Click save.
11. Open an administrator cmd instance on your device.
12. Navigate to the Remote Root Directory you specified before.
13. Install the following Java version on your device by following these installation instructions: <https://docs.aws.amazon.com/corretto/latest/corretto-17-ug/windows-7-install.html>. If you have WSL, this step is a little easier as you can just follow this guide: <https://docs.aws.amazon.com/corretto/latest/corretto-17-ug/generic-linux-install.html>. Note that this step is crucial as your device needs to have the same version of Java installed as the Jenkins instance.
14. Navigate to this page: <https://github.com/winsw/winsw/releases>
15. Scroll to the “Asset” section and download “WinSW-x64.exe”. This will be used to set up the agent as a Windows Service.
16. Copy the downloaded file into your Remote Root Directory and name it “jenkins-agent.exe”. This step is again crucial.
17. Create a new file in the directory named “jenkins-agent.xml”.
18. Open the xml file and put in the following code.

<service>

<id>jenkins</id>

<name>Jenkins</name>

<description>This service runs Jenkins continuous integration system.</description>

<env name="JENKINS\_HOME" value="%BASE%" />

<executable>java</executable>

<arguments>/////////////</arguments>

<log mode="roll" />

<onfailure action="restart" />

</service>

1. In the <arguments> section, replace the field with the agent information on Jenkins. This can be found by going to Jenkins, going to your agent and copying the line of code that should look similar to this:

-jar agent.jar -jnlpUrl http://<IP\_ADDRESS>:8080/computer/Test/jenkins-agent.jnlp -secret 2429fecda6d24516657f77e815c8d119aadbf4076133a0464d69e650d5a73abc -workDir ""

NOTE: DO NOT copy the “java” at the front of this text. Start with the -jar agent.jar….

The xml file should now look something like this.

<service>

<id>jenkins</id>

<name>Jenkins</name>

<description>This service runs Jenkins continuous integration system.</description>

<env name="JENKINS\_HOME" value="%BASE%" />

<executable>java</executable>

<arguments>-jar agent.jar -jnlpUrl http://<IP\_ADDRESS>:8080/computer/Test/jenkins-agent.jnlp -secret 2429fecda6d24516657f77e815c8d119aadbf4076133a0464d69e650d5a73abc -workDir ""</arguments>

<log mode="roll" />

<onfailure action="restart" />

</service>

1. Save the xml file.
2. Copy the line of code from Jenkins that is similar to this:

curl.exe -sO http://<IP\_ADDRESS>:8080/jnlpJars/agent.jar

1. Run it in your command prompt browser.
2. Now run:

jenkins-agent.exe install

1. Then run:

jenkins-agent.exe start

Now your agent should be up and running. To stop the agent at anytime run “jenkins-agent.exe stop”.

# A.4 Jenkins Pipeline with GitHub Repository

The following sections explain methods of creating pipelines in Jenkins that work with a GitHub repository. Note that the pipelines build upon each other.

## A.4.1 Basic Pipeline with Github Repository

Follow the steps to first log onto AWS and access Jenkins.

1. Open Jenkins.
2. On the home page select “New Item”.
3. Please type in your desired item name and “Pipeline” from the options below.
4. Select “OK”.
5. Go down to the pipeline section.
6. On the definition dropdown select “Pipeline script from SCM”.
7. On the SCM dropdown select “Git”.
8. Copy the GitHub repository URL into the field.
   1. If the GitHub repository is private, you will need to fill out some credentials.
9. Change the branch to the desired working branch.
10. Change the script path to the Jenkinsfile you made in the GitHub repository.
11. Click Save.

The pipeline will now be able to work in conjunction with your GitHub repository. This means that the Jenkinsfile can be updated on GitHub and the changes will be visible in Jenkins.

## A.4.2 Jenkins and webhooks

The following steps outline how to create a pipeline with a webhook for automatic scheduling of builds.

1. Open Jenkins.
2. On the home page select “New Item”.
3. Please type in your desired item name and “Pipeline” from the options below.
4. Select “OK”.
5. Go down to the pipeline section.
6. On the definition dropdown select “Pipeline script from SCM”.
7. On the SCM dropdown select “Git”.
8. Copy the GitHub repository URL into the field.
   1. If the GitHub repository is private, you will need to fill out some credentials.
9. Change the branch to the desired working branch.
10. Change the script path to the Jenkinsfile you made in the Github Repo.
11. Click Save.
12. Navigate to the GitHub Repo.
13. Click settings (note if you are trying to do this on the shared repository you will need admin permissions).
14. Click hooks on the left-hand side menu.
15. Copy and paste the following in the webhook address: http://<IP\_Address/github-webhook/ (The /github-webhook/ bit at the end is crucial as without it, Jenkins will redirect to the wrong place).
16. Change the content type to application/JSON.
17. Keep everything else as the default values.
18. Click save.
19. Now every time a push is made to that GitHub repository, the pipeline will be built.

## A.4.3 Jenkins Merge test

The following pipeline performs the following tasks”

* Create a branch.
* Modify a date file with the value of the current date.
* Test the code runs.
* Commit the code.
* Merge the branch.

The pipeline script is outlined below.

pipeline {

agent any

stages {

stage('Create Branch') {

steps {

sh '''

if git rev-parse --verify feature/update-date >/dev/null 2>&1; then

git checkout feature/update-date

else

git checkout -b feature/update-date

fi

'''

}

}

stage('Create or Update date.py') {

steps {

sh '''

echo "print('$(date)')" > date.py

'''

}

}

stage('Run Test on main.py') {

steps {

sh '''

echo "$SUDO\_PASS" | sudo -S yum update

echo "$SUDO\_PASS" | sudo -S yum install -y python3

python3 date.py

'''

}

}

stage('Commit Changes') {

steps {

// Add date.py to the staging area and commit the changes

sh '''

git add date.py

git commit -m "Update date.py with current date"

'''

}

}

stage('Merge Branch') {

steps {

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

sh '''

git config credential.helper 'store --file=.git/credentials'

echo "https://${GIT\_USERNAME}:${GIT\_PASSWORD}@github.cs.adelaide.edu.au" > .git/credentials

git checkout main

git pull origin main

git merge feature/update-date

git push origin main

'''

}

}

}

}

}

# A.5 Jenkins with Snyk

## A.5.1 Snyk Overview

Snyk is a software security platform and a set of related tools designed to help developers and organisations identify and remediate security vulnerabilities in their software applications and open-source dependencies. It is primarily focused on helping developers and DevOps teams secure their code and prevent security issues from making their way into the software development life cycle. We chose to use Snyk as it is extremely versatile in terms of code and was easy and quick to integrate into our pipeline relative to other SAST tools.

## A.5.2 Snyk Pipeline

The following steps outline how to integrate Snyk with Jenkins.

* + - 1. Download Snyk Security Plugin from available plugins.
      2. Navigate into tools in manage Jenkins to the bottom, set the name as “Snyk Security”
      3. Get Snyk auth token from (https://app.snyk.io/account).
      4. Navigate into Credentials in manage Jenkins to add new credential  
         (name: Snyk API token, Token is snyk auth token, self name the ID, and description).
      5. Create your pipeline, then click on the Pipeline Syntax, in snippet generator. Then choose snykSecurity: Invoke Snyk Security task, then choose your Snyk API token set in step 4(leave target file, Organisation project name as blank). Then generate, copy the whole code, and change 'Please define a Snyk installation in the Jenkins Global Tool Configuration. This task will not run without a Snyk installation.' to ‘Snyk Security’
      6. We now need to install Snyk on our node by first opening the console/terminal of the node we are using.
      7. Run the following (For Linux)

sudo apt update

sudo apt install –y node.js

sudo npm install –g snyk

For Mac:

brew update

/bin/bash -c "$(curl –fsSL <https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh>)"

brew install node

sudo npm install --global snyk

1. Confirm it has been updated by doing the following:

snyk –version

1. Authentical your snyk by running:

snyk auth <your api token>

1. Access the following page on Jenkins Manage Jenkins -> Tools.
2. Scroll until the Snyk section and add a Snyk installation with a name selected by you and the installation directory as what is printed out when you run “which snyk” on your nodes terminal (it will probably be ”/usr/local/bin/snyk”).
3. Make sure install automatically is left unticked.
4. Go to your nodes page.
5. Go to your nodes configuration and scroll until you see the “tool locations” check box and then check the box associated with it.
6. From the drop down that will appear select the snyk variant you set up in the tools section earlier.
7. Jenkins will now work the following code should run fine (Not it will not complete if vulnerability is identified by snyk but will still have ran a scan). Refer to Max\_Snyk for working implementation (don't run too many times though as api keys can only run so often).

A sample Jenkinsfile with Snyk is given below:

pipeline {

agent any

stages {

stage('Download'){

steps{

echo "downloading data"

git branch: 'main', credentialsId: 'uniGithub\_Hugh', url: '<https://github.cs.adelaide.edu.au/SEP-CD/CD1.git>'

}

}

stage('Build') {

steps {

echo 'Building...'

}

}

stage('Test') {

steps {

dir('testProjectCodes') {

echo 'Testing...'

snykSecurity snykInstallation: 'Snyk Security', snykTokenId: 'test'

}

}

}

stage('Deploy') {

steps {

echo 'Deploying...'

}

}

}

}

# A.6 Prometheus

## A.6.1 General Setup Overview

After installing the Prometheus plugin on Jenkins, it is configured so that data is collected every 10 minutes and made available at this URL: http://<IP Address>:8080/prometheus. This bit of Prometheus is set up in the Jenkins system. This was done by installing the Prometheus plugin, navigating to the system configuration settings and leaving all the defaults except for changing the data collected to every 10 minutes. At this point, Jenkins uses the Prometheus plugin to collect data and sends it to the address http://<IP Address>:8080/prometheus but nothing actually happens with it yet.

Prometheus is installed on the Jenkins instance of the AWS EC2 server. To access the console, click on the instance, click connect and then a terminal should pop up. This step is needed to make any changes to the configuration of Prometheus once it has been set up in docker, such as the scrape time and the web address.

Prometheus is configured by the prometheus.yml file which is located on the EC2 server at cd ~/prometheus-2.47.0.linux-amd64/prometheus.yml. To format the configuration file, there is an example file as given here:

<https://gist.github.com/darinpope/1c8422fb7512411760ccb2827d82613f>

The current setup of the configuration file is shown below:

global:

scrape\_interval: 15s

evaluation\_interval: 15s

scrape\_timeout: 10s

# Alertmanager configuration

alerting:

alertmanagers:

- static\_configs:

- targets:

# - alertmanager:9093

# Load rules once and periodically evaluate them according to the global 'evaluation\_interval'.

rule\_files:

# - "first\_rules.yml"

# - "second\_rules.yml"

# A scrape configuration containing exactly one endpoint to scrape:

# Here it's Prometheus itself.

scrape\_configs:

- job\_name: 'prometheus'

static\_configs:

- targets: ['<Current IP Address>:9090']

- job\_name: 'jenkins'

metrics\_path: /prometheus/

static\_configs:

- targets: ['<Current IP Address>:8080']

## A.6.2 How to Connect and Run Prometheus

Go to the AWS EC2 server and run the following commands.

sudo systemctl enable prometheus

sudo systemctl restart prometheus

If this doesn’t work (sudo systemctl status prometheus -> will tell you if it works) do this:

Change /etc/systemd/system/prometheus.service

[Unit]   
Description=Prometheus   
Wants=network-online.target   
After=network-online.target[Service]   
User=ec2-user  
Group=ec2-user  
Type=simple   
ExecStart=/usr/local/bin/prometheus \   
 --config.file /etc/prometheus/prometheus.yml \   
 --storage.tsdb.path /var/lib/prometheus/ \   
 --web.console.templates=/etc/prometheus/consoles \   
 --web.console.libraries=/etc/prometheus/console\_libraries[Install]   
WantedBy=multi-user.target

You may need to run this code after setting it:

sudo mkdir -p /etc/prometheus/consoles

sudo mkdir -p /etc/prometheus/console\_libraries

sudo mkdir -p /var/lib/prometheus/

sudo chown -R ec2-user:ec2-user /etc/prometheus/

sudo chown -R ec2-user:ec2-user /var/lib/prometheus/

And you will need to run this:

systemctl daemon-reload

and then try starting Prometheus again with

sudo systemctl start prometheus

These commands will not run promethius in the background only use them if you can’t get that to work:

cd ~/prometheus-2.47.0.linux-amd64/

then

./prometheus --config.file=prometheus.yml

To connect to Prometheus, click on the following link: http://<IP Address>:9090/

## A.6.3 If Prometheus Crashes

sudo systemctl restart Prometheus

If this doesn’t work you will need to fix whatever error there is with the file which allows it to run at boot /etc/systemd/system/prometheus.service or the config file: /etc/prometheus/prometheus.yml which should say something similar to below:

GNU nano 5.8 /etc/prometheus/prometheus.yml

global:

scrape\_interval: 10m

scrape\_configs:

- job\_name: 'prometheus'

static\_configs:

- targets: ['localhost:9090']

- job\_name: 'jenkins'

static\_configs:

- targets: ['localhost:8080']

metrics\_path: '/prometheus/jenkins'

basic\_auth:

username: <Jenkins username>

password: <Jenkins password>

If restart works but Prometheus is not running at boot run this line:

sudo systemctl enable prometheus)

If pushgateway is working add this:

- job\_name: 'pushgateway'

static\_configs:

- targets: [‘localhost:9091']

(changes to the service file can be loaded with: sudo systemctl daemon-reload)

## A.6.4 Chaning the yml file

If you change the YML file you will need to kill the current process and re run Prometheus to have the most up to date information. Do the following:

lsof -i 9090

Then copy the PID of the current Prometheus process and run the following command.

sudo kill -9 <pid>

You can now restart Prometheus with the updated yml file.

# A.7 Grafana

## A.7.1 Overview

Grafana is an open-source platform for monitoring and observability that is widely used in the field of DevOps and system administration. It provides tools for creating, analysing, and visualising metrics, logs, and other data from various sources in real-time. In our project specifically, we use Grafana to track and display the pipeline performance based on the requested metrics.

## A.7.2 Install

To install:

wget https://dl.grafana.com/enterprise/release/grafana-enterprise-10.0.0-1.x86\_64.rpm

sudo rpm -Uvh grafana-enterprise-10.0.0-1.x86\_64.rpm

May also have to run: sudo yum update

To ensure that Grafana starts automatically when your system boots up, enable it as a system service.

sudo systemctl daemon-reload

sudo systemctl enable grafana-server

Start Grafana: First, make sure that Grafana is installed and available on your system. You can start it using the command-line interface.

sudo systemctl start grafana-server

Check the Status: You can check the status of Grafana to ensure it's running without any issues.

sudo systemctl status grafana-server

This command will display information about the service, including its status (active/running), process ID, and more.

## A.7.3 Accessing Grafana

By default, Grafana runs on port 3000. You can access it in your web browser by typing in the IP address or domain name of your server followed by the port number. For example:

http://<IP\_Address>:3000

If you want to run Grafana on a different port, you'll need to configure it in the Grafana configuration file.

Change the Port (Optional): If you want to run Grafana on a different port, you can edit the Grafana configuration file. The configuration file is usually located at /etc/grafana/grafana.ini. You can use a text editor to open it.

sudo nano /etc/grafana/grafana.ini

Look for the [server] section in the configuration file, and change the http\_port option to the desired port number:

[server]

http\_port = 8080 # Change this to your desired port number

Save the file and exit the text editor.

Restart Grafana: After changing the port, restart Grafana for the changes to take effect.

Copy code

sudo systemctl restart grafana-server

Now, Grafana should be running on the port you specified in the configuration file. You can access it in your web browser using the new port number, like http://<IP\_Address>:8080 if you set it to port 8080.

## A.7.4 Password Reset

To reset the Grafana password:

sudo grafana-cli admin reset-admin-password admin

Resets the password so user and password is both admin.

## A.7.5 If Grafana Crashes

sudo systemctl restart grafana-server

# A.8 If a program crashes

Perform the following is a program on the EC2 instance crash but not the entire server. Below is an example for Grafana.

Replace port and app name.

sudo lsof -i :9090

sudo kill <PID>

sudo systemctl start grafana-server

# A.9 Metrics

There will be two sets of metrics that we will capture in this project.

* Delivery speed-related metrics
* Security assessment metrics

## A.9.1 Speed related metrics

**Deployment Frequency**

Definition: This metric measures how often code is deployed to staging or production env.

Importance: A higher deployment frequency is indicative of a mature CI/CD pipeline, where changes are smaller and more manageable. It can reflect an organization's ability to deliver features, fixes, and updates to users at a rapid pace.

How to Measure: Track the number of commits over a specific period (e.g., daily, weekly, monthly) that is delivered to the staging or production env.

**Lead Time for Changes**

Definition: This is the time it takes for a commit to be deployed to staging or production env. once it has been merged into the main branch.

Importance: A shorter lead time indicates a more efficient pipeline, allowing for faster feedback and quicker delivery of features or fixes to users.

How to Measure: Calculate the time difference between code being merged and it being successfully deployed in staging or production env.

**Test Pass Rate**

Definition: The percentage of automated tests that pass during the CI phase.

Importance: This metric gives an insight into code quality and the effectiveness of the testing process. A high pass rate means the codebase is stable, while a sudden drop might indicate a problematic commit or an issue with the testing environment.

How to Measure: Divide the number of tests that pass by the total number of tests run in the CI server. Monitor this metric per build or over a set period.

## A.9.2 Security assessment metrics

**Number of vulnerabilities based on CVSS**

A screen shot of a chart

Description automatically generated

**SAST Tool runtime:** The time taken to run the tool after each commit.

# A.10 Snyk Monitored by Prometheus

Hugh’s Snyk account Tokens:

Org ID: 0971f52f-bca0-404f-97a7-6e79f507e0ef

Snyk Api token: c1b1c438-3274-4697-adbd-02576a16cbf2

Project ID: cab23c5a-1ad9-4a85-b502-d77bdbe3317e

Method 1

Ended Up with snyk is not entitled to api access, needs to upgrade plan

1. Use snyk\_exporter (The Snyk Exporter has been archived as it is no longer actively maintained.)

<https://github.com/lunar/snyk_exporter>

1. Use node exporter to expose the report the website

https://github.com/prometheus/node\_exporter

1. (Recommend working on this one ) Snyk scraper method is near the end of the page, expose the data to the website with port number 9532 (super easy on this one, 9532 is normally used for snyk exporter)

<https://snyk.io/blog/vulnerability-monitoring-with-snyk-prometheus-and-grafana/>

docker run -p9532:9532 quay.io/lunarway/snyk\_exporter --snyk.api-token [SnykAPIToken] --log.level="info" --snyk.organization=[SnykOrganizationID]

Method 2

sh 'snyk test --json > snyk-results.json'

1. Manually install the snyk, and then use the above command in the project folder to run the test to get the json file. (Got permission denied while installing the snyk on my computer, it is worth to try, since the Jenkinsfile in the visualWebCodes can install npm)

# A.11 Installing pushgateway for Prometheus

The contents of this appendix outline how to install a pushgateway for Prometheus.

1. Download the latest version of Pushgateway for Linux

wget https://github.com/prometheus/pushgateway/releases/download/v1.4.2/pushgateway-1.4.2.linux-amd64.tar.gz

1. Extract the downloaded tarball:

tar xvfz pushgateway-1.4.2.linux-amd64.tar.gz

1. Move the Pushgateway binary to a suitable location:

sudo mv pushgateway-1.4.2.linux-amd64/pushgateway /usr/local/bin/

1. Create a systemd service file for Pushgateway:

sudo tee /etc/systemd/system/pushgateway.service <<EOF

[Unit]

Description=Prometheus Pushgateway

Wants=network-online.target

After=network-online.target

[Service]

User=ec2-user

ExecStart=/usr/local/bin/pushgateway

[Install]

WantedBy=multi-user.target

EOF

1. Reload systemd to recognize the new service

sudo systemctl daemon-reload

1. Enable the service to start on boot

sudo systemctl enable pushgateway

1. Start the Pushgateway service

sudo systemctl start pushgateway

1. Add 9091 to security group
2. Add the following to Prometheus.yml:

scrape\_configs:

- job\_name: 'pushgateway'

static\_configs:

- targets: [localhost:9091']

# A.12 Successful SAST tool PIPELINE Prometheus GATEWAY

This code is designed to give a failing CVSS result so exit 0 is used to make the test pass. On the product, exit 0 will not be used and a bad CVSS score will result in a failure, however as this is designed to always fail there is no point. Furthermore, this code assumes a Linux based agent and so the code will need to be adapted for different operating systems.

pipeline {

agent {

label 'my-agent-label'

}

environment {

SNYK\_TOKEN = '50b15e06-0453-4c5c-ad25-942a6808536e'

}

stages {

/\*

stage('Sync System Clock') {

steps {

script {

if (isUnix() && sh(script: 'command -v apt', returnStatus: true) == 0) {

sh '''

sudo apt install -y ntp

sudo systemctl start ntp

sudo systemctl enable ntp

sleep 10 # Give it a few seconds to sync the time

'''

}

}

}

}

stage('Check Time Sync') {

steps {

script {

if (isUnix() && sh(script: 'command -v apt', returnStatus: true) == 0) {

def timeOffset = sh(script: "ntpq -c rv | grep -o 'offset=[-0-9.]\*' | cut -d= -f2", returnStdout: true).trim()

if (Math.abs(Float.parseFloat(timeOffset)) > 100) { // Adjust the threshold as needed

error "System clock is not synchronized. Offset is ${timeOffset}ms."

}

}

}

}

}

\*/

stage('apt update if ubuntu') {

steps {

script {

if (isUnix() && sh(script: 'command -v apt', returnStatus: true) == 0) {

timeout(time: 1, unit: 'MINUTES') {

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

sh '''

sudo apt update

'''

}

}

}

}

}

}

stage('Setup Environment') {

steps {

retry(3) {

script {

// Check if the system is Ubuntu

if (isUnix() && sh(script: 'command -v apt', returnStatus: true) == 0) {

echo "Ubuntu detected. Installing Docker using apt."

sh '''

sudo apt install -y apt-transport-https ca-certificates curl software-properties-common

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable"

sudo apt update

sudo apt install -y docker-ce

'''

} else if (sh(script: 'command -v yum', returnStatus: true) == 0) {

echo "Amazon Linux detected. Installing Docker using yum."

sh '''

sudo yum update -y

sudo yum install -y docker

sudo systemctl start docker

sudo systemctl enable docker

'''

} else {

error "Unsupported OS. Cannot install Docker."

}

}

}

}

}

stage('Check Docker Status') {

steps {

script {

// Check if Docker is running

def status = sh(script: 'sudo docker info', returnStatus: true)

if (status != 0) {

echo "Docker is not running. Attempting to start Docker."

sh 'sudo dockerd &'

sleep 10 // Give Docker a few seconds to start

}

}

}

}

/\* stage('Install Docker') {

steps {

withCredentials([string(credentialsId: 'SUDO\_PASS', variable: 'SUDO\_PASS')]) {

sh '''

echo $SUDO\_PASS | sudo -S yum update -y

echo $SUDO\_PASS | sudo -S yum install -y docker

echo $SUDO\_PASS | sudo -S systemctl start docker

echo $SUDO\_PASS | sudo -S systemctl enable docker

'''

}

}

}

\*/

stage('Setup Docker Environment') {

steps {

withCredentials([string(credentialsId: 'SUDO\_PASS', variable: 'SUDO\_PASS')]) {

script {

def dockerfileContent = """

FROM node:14

RUN npm install -g snyk@1.1226.0

"""

writeFile file: 'Dockerfile', text: dockerfileContent

sh 'echo $SUDO\_PASS | sudo -S docker build -t custom-node-snyk-image -f Dockerfile .'

}

}

}

}

stage('Create Branch') {

steps {

sh '''

if git rev-parse --verify feature/update-date >/dev/null 2>&1; then

git checkout feature/update-date

else

git checkout -b feature/update-date

fi

'''

}

}

stage('Create or Update date.py') {

steps {

sh '''

echo "print('$(date)')" > date.py

'''

}

}

stage('Create or Update Files in scannable-directory') {

steps {

sh '''

if [ ! -d "scannable-directory" ]; then

mkdir -p scannable-directory

fi

echo 'print(\"'\"$(date)\"'\")' > scannable-directory/date.py

echo 'print("Hello, World!")' > scannable-directory/hello.py

echo 'password = "weak\_password"' > scannable-directory/hello.py

'''

}

}

stage('Create Scannable Directory') {

steps {

sh '''

if [ ! -d "scannable-directory" ]; then

mkdir -p scannable-directory

fi

cp date.py scannable-directory/

docker run --rm -v $(pwd):/workspace -w /workspace custom-node-snyk-image sh -c "

cd scannable-directory

cat <<EOF > package.json

{

\\"dependencies\\": {

\\"lodash\\": \\"4.17.15\\"

}

}

"

'''

}

}

stage('Debug: Print package.json + npm install') {

steps {

sh '''

docker run --rm -v $(pwd):/workspace -w /workspace custom-node-snyk-image sh -c "

cd scannable-directory

npm install

cat package.json

"

'''

}

}

/\*

stage('Run Snyk Test') {

steps {

sh '''

docker run --rm -v $(pwd):/workspace -w /workspace -e SNYK\_TOKEN=${SNYK\_TOKEN} custom-node-snyk-image sh -c "

cd scannable-directory

snyk auth ${SNYK\_TOKEN}

snyk test --all-projects --debug

exit 0

"

'''

}

}

\*/

// This will pass regardless if flaws are identified

// if you remove exit 0 it will fail because of the security vulnerabilities, when we are testing this on a proper product

// you will want it to fail on that result

stage('Run Snyk Test') {

steps {

script {

// Capture the start time

def startTime = System.currentTimeMillis()

sh '''

docker run --rm -v $(pwd):/workspace -w /workspace -e SNYK\_TOKEN=${SNYK\_TOKEN} custom-node-snyk-image sh -c "

cd scannable-directory

snyk auth ${SNYK\_TOKEN}

snyk test --all-projects --debug > snyk\_output.txt

exit 0

"

'''

// Calculate the duration in seconds

def duration = (System.currentTimeMillis() - startTime) / 1000

echo "Snyk scan took ${duration} seconds"

// Store the duration in an environment variable for later use

env.SNYK\_SCAN\_DURATION = "${duration}"

}

}

}

stage('Display Snyk Output') {

steps {

sh '''

cat scannable-directory/snyk\_test\_output.txt

'''

}

}

stage('Push Results to Pushgateway') {

steps {

script {

// Extract the IP or hostname from the JENKINS\_URL

def jenkinsURL = new URL(env.JENKINS\_URL)

def jenkinsHost = jenkinsURL.getHost()

// Parse the snyk\_output.txt file to extract severity ratings

def ratings = sh(script: """

grep -oE '\\[(Low|Medium|High|Critical) Severity\\]' scannable-directory/snyk\_output.txt | cut -d'[' -f2 | cut -d' ' -f1 | sort | uniq -c

""", returnStdout: true).trim()

// Initialize counters for each severity

def totalVulnerabilities = ['Low': 0, 'Medium': 0, 'High': 0, 'Critical': 0]

// Update the counters based on the parsed ratings

ratings.split("\n").each { line ->

def (count, rating) = line.split()

totalVulnerabilities[rating] += count.toInteger()

}

// Initialize a global counter for metric pushes

def globalCounter = 0

// Push individual and total counts for each severity to Pushgateway

totalVulnerabilities.each { rating, count ->

echo "Pushing ${rating} vulnerabilities count: ${count}"

sh """

echo "snyk\_vulnerabilities\_${globalCounter}{severity=\\"${rating}\\", instance=\\"${env.NODE\_NAME}\\"} ${count}" | curl --data-binary @- http://${jenkinsHost}:9091/metrics/job/snyk\_scan

"""

globalCounter++

}

// Push the Snyk scan duration to Pushgateway

echo "Pushing Snyk scan duration: ${env.SNYK\_SCAN\_DURATION} seconds"

sh """

echo "snyk\_scan\_duration\_seconds\_${globalCounter}{instance=\\"${env.NODE\_NAME}\\"} ${env.SNYK\_SCAN\_DURATION}" | curl --data-binary @- http://${jenkinsHost}:9091/metrics/job/snyk\_scan

"""

globalCounter++

// Extract the names of all the issues associated with their severity level and push them to Pushgateway

['Low', 'Medium', 'High', 'Critical'].each { severity ->

def issues = sh(script: """

grep -oE "✗ [^\\\\[]+ \\\\[${severity} Severity\\\\]" scannable-directory/snyk\_output.txt | awk -F '✗ ' '{print \$2}' | awk -F ' \\\\[${severity}' '{print \$1}' | tr '\n' ','

""", returnStdout: true).trim()

if (issues) {

echo "Pushing ${severity} vulnerabilities: ${issues}"

sh """

echo "snyk\_issues\_${globalCounter}{severity=\\"${severity}\\", issues=\\"${issues}\\", instance=\\"${env.NODE\_NAME}\\"} 1" | curl --data-binary @- http://${jenkinsHost}:9091/metrics/job/snyk\_scan

"""

globalCounter++

}

}

}

}

}

stage('Merge Branch') {

steps {

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

sh '''

git config credential.helper 'store --file=.git/credentials'

echo "https://${GIT\_USERNAME}:${GIT\_PASSWORD}@github.cs.adelaide.edu.au" > .git/credentials

git checkout main

git pull origin main

git merge feature/update-date

git push origin main

'''

}

}

}

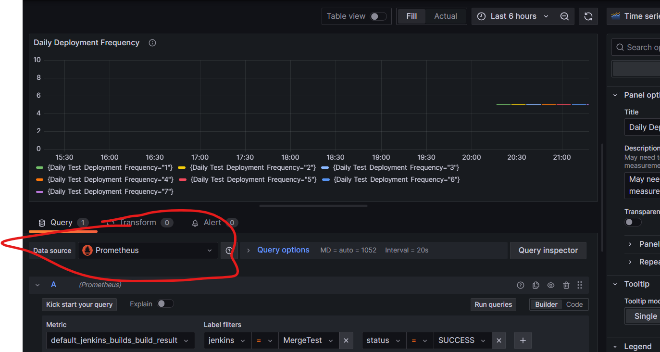
}

}

# A.13 Implementation of Speed Metrics

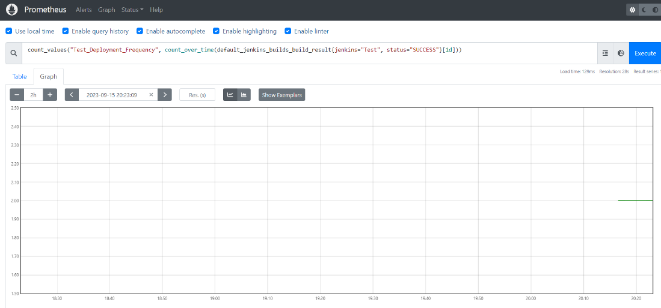
## A.13.1 How to Create a Grafana Dashboard

* 1. Login to Grafana using http://<IP\_Address>:3000
  2. Navigate to the dashboard section.
  3. Create a new dashboard
  4. Name it
  5. Create a visualisation and ensure Prometheus is selected as shown below.

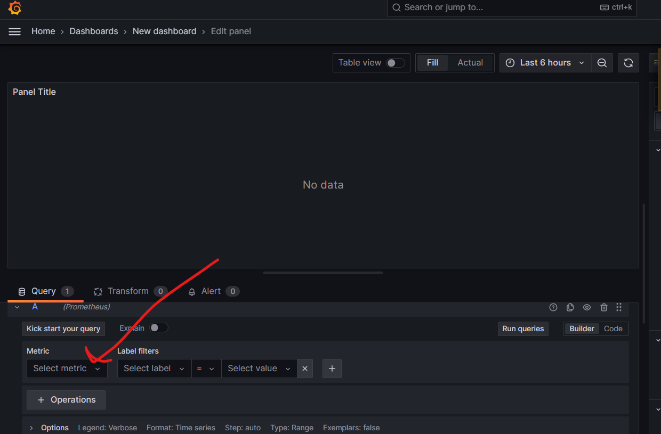


* 1. Write code to call upon the data which is stored here: http://<IP Address>:8080/jenkins/prometheus

Prometheus code can be tested at: http://<IP Address>:9090/graph. This offers a friendly UI that can help is creating queries.



1. Paste the working code into metrics on Grafana



## A.13.2 Code for deployment frequency

The Prometheus code for the deployment frequency is shown below.

git\_deployment\_frequency\_daily

This is obtained through gateway using this code:

stage('Send Deployment Frequency') {,

steps {

script {

// Count the number of merges to the main branch

def mergeCount = sh(script: 'git log --oneline --merges --since="1 day ago" origin/main | wc -l', returnStdout: true).trim()

// Send this count to gateway

sh """

echo 'git\_deployment\_frequency\_daily {mergeCount="${mergeCount}"} ${mergeCount}' | curl --data-binary @- http://${JENKINS\_HOST}:9091/metrics/job/git\_metrics

"""

}

}

A graph on a black background

Description automatically generated

## A.13.3 Code for test pass rate

The Prometheus code for the test pass rate is shown below.

sum(count\_over\_time(default\_jenkins\_builds\_build\_result{jenkins\_job="MergeTest", status="SUCCESS"}[1d]))/sum(count\_over\_time(default\_jenkins\_builds\_build\_result{jenkins\_job="MergeTest"}[1d]))

A screenshot of a computer

Description automatically generated

## A.13.4 Code For Lead Time for Changes

The Prometheus code for lead time for changes is shown below.

default\_jenkins\_builds\_last\_build\_duration\_milliseconds{jenkins\_job="MergeTest"}/1000

# A.14 Extraction of SAST Metrics

The following pipeline script outlines how the Snyk metrics are extracted from the pipeline and pushed to the gateway.

stage('Push Results to Pushgateway') {,

steps {

script {

// Parse the snyk\_output.txt file to extract severity ratings

def ratings = sh(script: """

grep -oE '\\[(Low|Medium|High|Critical) Severity\\]' PythonPlantsVsZombies/snyk\_test\_output.txt | cut -d'[' -f2 | cut -d' ' -f1 | sort | uniq -c

""", returnStdout: true).trim()

// Initialize counters for each severity

def totalVulnerabilities = ['Low': 0, 'Medium': 0, 'High': 0, 'Critical': 0]

// Update the counters based on the parsed ratings

ratings.split("\n").each { line ->

def (count, rating) = line.split()

totalVulnerabilities[rating] += count.toInteger()

}

// Initialize a global counter for metric pushes

def globalCounter = 0

// Push individual and total counts for each severity to Pushgateway

totalVulnerabilities.each { rating, count ->

echo "Pushing ${rating} vulnerabilities count: ${count}"

sh """

echo "snyk\_vulnerabilities\_${globalCounter}{severity=\\"${rating}\\", instance=\\"${env.NODE\_NAME}\\"} ${count}" | curl --data-binary @- http://${JENKINS\_HOST}:9091/metrics/job/snyk\_scan

"""

globalCounter++

}

// Push the Snyk scan duration to Pushgateway

echo "Pushing Snyk scan duration: ${env.SNYK\_SCAN\_DURATION} seconds"

sh """

echo "snyk\_scan\_duration\_seconds\_${globalCounter}{instance=\\"${env.NODE\_NAME}\\"} ${env.SNYK\_SCAN\_DURATION}" | curl --data-binary @- http://${JENKINS\_HOST}:9091/metrics/job/snyk\_scan

"""

globalCounter++

// Extract the names of all the issues associated with their severity level and push them to Pushgateway

['Low', 'Medium', 'High', 'Critical'].each { severity ->

def issues = sh(script: """

grep -oE "✗ [^\\\\[]+ \\\\[${severity} Severity\\\\]" PythonPlantsVsZombies/snyk\_test\_output.txt | awk -F '✗ ' '{print \$2}' | awk -F ' \\\\[${severity}' '{print \$1}' | tr '\n' ','

""", returnStdout: true).trim()

if (issues) {

echo "Pushing ${severity} vulnerabilities: ${issues}"

sh """

echo "snyk\_issues\_${globalCounter}{severity=\\"${severity}\\", issues=\\"${issues}\\", instance=\\"${env.NODE\_NAME}\\"} 1" | curl --data-binary @- http://${JENKINS\_HOST}:9091/metrics/job/snyk\_scan

"""

globalCounter++

}

}

}

}

}

## A.14.1 SAST tool runtime code

Prometheus code:

snyk\_scan\_duration\_seconds

A screen shot of a graph

Description automatically generated

## A.14.2 Number of vulnerabilities based on CVSS:

Pipeline code for defining vulnerabilities.

(snyk\_vulnerabilities\_0 = low, snyk\_vulnerabilities\_1 = medium, snyk\_vulnerabilities\_2= high, snyk\_vulnerabilities\_3 =critical)

Prometheus code:

snyk\_vulnerabilities\_0

or

snyk\_vulnerabilities\_1

or

snyk\_vulnerabilities\_2

or

snyk\_vulnerabilities\_3

# A.15 Simulation

The following image depict the metrics that were measured during the PlantsVsZombies simulation.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a phone

Description automatically generated

A screenshot of a video

Description automatically generated

A screenshot of a computer

Description automatically generated

## A.15.1 Finalised Metrics for Simulation

The following sections outline the code for the metrics calculations as part of the simulation.

### A.15.1.1 **Weighted Vulnerability Score**

The weighted metrics score is calculated according giving cvss vulnerabilities the following scores:

def weightMap = [

'Low': 1,

'Medium': 2,

'High': 5,

'Critical': 10

].

Prometheus query:

Synk\_total\_vulnerabilities\_weighted

A screenshot of a computer

Description automatically generated

Code: included in the jenkinsfile in the code section below, see push gateway section of the jenkinsfile for custom code that captures this.

### A.15.1.2 **Deployment Frequency daily from PlantsVsZombies**

Prometheus query

git\_commit\_frequency\_daily\_PythonPlantsVsZombies

Code: included in the jenkinsfile in the code section below, see push gateway section of the jenkinsfile for custom code that captures this.

A screenshot of a computer

Description automatically generated

### A.15.1.3 **Deployment Frequency Vs Weighted Synk Score**

(change plot to XY plot)

Grafana Query X-axis: git\_commit\_frequency\_daily\_PythonPlantsVsZombies

Grafana Query Y-axis: Synk\_total\_vulnerabilities\_weighted

A screenshot of a computer

Description automatically generated

Transformations:

A black screen with a black background

Description automatically generated with medium confidence

(transformations must be applied in the order as above or graph will not work)

### A.15.1.4 **Total Vulnerabilities**

Pipeline code:

(snyk\_vulnerabilities\_0

= low, snyk\_vulnerabilities\_1 = medium, snyk\_vulnerabilities\_2= high, snyk\_vulnerabilities\_3 =critical)

Grafana Query:

snyk\_vulnerabilities\_0

or

snyk\_vulnerabilities\_1

or

snyk\_vulnerabilities\_2

or

snyk\_vulnerabilities\_3

Code: included in the jenkinsfile in the code section below, see push gateway section of the jenkinsfile for custom code that captures this.

A screen shot of a computer

Description automatically generated

### A.15.1.5 **SAST tool runtime**

Prometheus query:

snyk\_scan\_duration\_seconds

A screenshot of a computer

Description automatically generated

Code: included in the jenkinsfile in the code section below, see push gateway section of the jenkinsfile for custom code that captures this.

### A.15.1.5 **Test Pass Rate**

Prometheus code:

sum(count\_over\_time(default\_jenkins\_builds\_build\_result{jenkins\_job="Simulation", status="SUCCESS"}[1d]))

/

sum(count\_over\_time(default\_jenkins\_builds\_build\_result{jenkins\_job="Simulation"}[1d]))

A graph on a black background

Description automatically generated

### A.15.1.6 **Lead Time to Change**

(Still uses Prometheus Jenkins plugin instead of custom code and Prometheus gateway, results may need further investigation)

Prometheus query:

default\_jenkins\_builds\_last\_build\_duration\_milliseconds{jenkins\_job="Simulation"} / 1000

A screen shot of a computer

Description automatically generated

# A.16 Final Pipeline

This appendix contains the Jenkins script for the final pipeline. The pipeline is a culmination of all the work that has been completed previously.

pipeline {

agent any

environment {

SNYK\_TOKEN = '50b15e06-0453-4c5c-ad25-942a6808536e'

// Extract the IP or hostname from the JENKINS\_URL

JENKINS\_HOST = new URL(env.JENKINS\_URL).getHost()

}

stages {

stage('Determine Agent') {

steps {

script {

if (isUnix()) {

if (sh(script: 'command -v apt', returnStatus: true) == 0) {

env.AGENT\_LABEL = 'my-agent-label'

} else if (sh(script: 'command -v brew', returnStatus: true) == 0) {

env.AGENT\_LABEL = 'NodeHugh'

} else {

env.AGENT\_LABEL = 'unknown-agent'

}

} else {

env.AGENT\_LABEL = 'unknown-agent'

}

}

}

}

// should add clear caches

// sudo rm -r /var/jenkins/workspaces

stage('update package manager') {

steps {

script {

if (isUnix()) {

// Check if the system is Ubuntu

def osType = sh(script: 'uname', returnStdout: true).trim()

if (osType == "Darwin") {

timeout(time: 1, unit: 'MINUTES') {

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

sh 'brew update'

}

}

}

else if (sh(script: 'command -v apt', returnStatus: true) == 0) {

timeout(time: 1, unit: 'MINUTES') {

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

sh 'sudo apt update'

}

}

}

else {

error "Unsupported operating system"

}

} else {

error "Unsupported operating system"

}

}

}

}

stage('Setup Environment') {

steps{

// retry(3) {

script {

// Check if the system is Ubuntu

def installDocker = {

if (sh(script: 'command -v apt', returnStatus: true) == 0) {

// Check if Docker is installed

if (sh(script: 'docker --version', returnStatus: true) != 0) {

echo "Docker is not installed. Installing Docker using apt."

echo "Ubuntu detected. Installing Docker using apt."

sh '''

sudo apt install -y apt-transport-https ca-certificates curl software-properties-common

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable"

sudo apt update

sudo apt install -y docker-ce

sudo service docker start

'''

}

} else if (sh(script: 'command -v yum', returnStatus: true) == 0) {

echo "Amazon Linux detected. Installing Docker using yum."

sh '''

sudo yum update -y

sudo yum install -y docker

sudo systemctl start docker

sudo systemctl enable docker

'''

}else {

// error "Unsupported OS. Cannot install Docker."

echo "Unsupported OS. Cannot install Docker."

}

}

def osType = sh(script: 'uname', returnStdout: true).trim()

if (osType == "Darwin") {

def brewInstalled = sh(script: 'command -v brew', returnStatus: true) == 0

if (!brewInstalled) {

echo "Homebrew not detected. Installing Homebrew first."

sh '''

/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install.sh)"

'''

}

if (sh(script: 'command -v brew', returnStatus: true) == 0) {

echo "Homebrew installed. Installing Docker using brew."

sh '''

brew install --cask docker

'''

}

else {

echo "Failed to install Homebrew. You are not using MacOs."

}

}else{

retry(3, { installDocker() })

}

}

}

}

stage('Configure Docker Builder') {

steps {

script {

sh 'export DOCKER\_CLI\_ACI=0'

}

}

}

stage('Check Docker Status') {

steps {

script {

// Check if Docker is running

echo "Check Docker Status"

def status = sh(script: 'sudo docker info', returnStatus: true)

if (status != 0) {

def osType = sh(script: 'uname', returnStdout: true).trim()

if (osType == "Darwin") {

echo "Docker is not running. Attempting to start Docker Desktop."

// Open Docker Desktop using AppleScript

sh 'osascript -e \'tell application "Docker" to activate\''

sleep 10 // Give Docker a few seconds to start

}

else{

echo "Docker is not running. Attempting to start Docker."

sh 'sudo dockerd &'

sleep 10 // Give Docker a few seconds to start

}

}

}

}

}

// Commented to notice that this should be changed, but not needed for now

stage('Setup Docker Environment for Python') {

steps {

script {

def osType = sh(script: 'uname', returnStdout: true).trim()

if (osType == "Darwin") {

def pythonCheck = sh(script: 'python3 --version', returnStatus: true)

if (pythonCheck != 0) {

echo 'Python not found, installing...'

sh

'''

brew install python3

'''

} else {

echo 'Python already installed.'

}

def pipCheck = sh(script: 'pip --version', returnStatus: true)

if (pipCheck != 0) {

echo 'pip3 not found, installing...'

sh '''

curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py

python3 get-pip.py

'''

} else {

echo 'pip3 already installed.'

}

} else {

// Linux setup - You can keep your existing Linux Docker setup here

def dockerfileContent = """

# Start with a Node.js base image

FROM node:16.14 AS node-base

# Clear npm cache

RUN npm cache clean -f

# Install required global npm packages

RUN npm install -g pnpm snyk@1.1229.0

# Switch to a Python image for Python-specific tasks

FROM python:3.11

# Install nodejs and npm to install snyk

RUN apt-get update && apt-get install -y python3-venv nodejs npm

# Install snyk using npm

RUN npm install -g snyk@1.1229.0

# Install venv module

RUN apt-get update && apt-get install -y python3-venv

# Set up a virtual environment

RUN python3 -m venv /workspace/PythonPlantsVsZombies/myenv

# Activate the virtual environment and install Python dependencies

COPY PythonPlantsVsZombies/Requirements.txt /workspace/PythonPlantsVsZombies/Requirements.txt

RUN /bin/bash -c "source /workspace/PythonPlantsVsZombies/myenv/bin/activate && pip install -r /workspace/PythonPlantsVsZombies/Requirements.txt"

"""

writeFile file: 'Dockerfile', text: dockerfileContent

sh 'echo $SUDO\_PASS | sudo -S docker build -t custom-python-snyk-image -f Dockerfile .'

}

}

}

}

stage('Build Docker Image') {

steps {

script {

def osType = sh(script: 'uname', returnStdout: true).trim()

if (osType != "Darwin") {

sh 'docker build -t custom-python-snyk-image -f Dockerfile .'

}

}

}

}

stage('Check PythonPlantsVsZombies in Docker') {

steps {

script {

def osType = sh(script: 'uname', returnStdout: true).trim()

if (osType != "Darwin") {

// Run a Docker command to check if the PythonPlantsVsZombies directory exists inside the container

def dirExists = sh(script: 'docker run --rm -v $(pwd):/workspace -w /workspace custom-python-snyk-image sh -c "[ -d PythonPlantsVsZombies ] && echo exists || echo not exists"', returnStdout: true).trim()

if (dirExists == "exists") {

echo "PythonPlantsVsZombies directory exists inside the Docker container."

// Print the file structure of the directory inside the Docker container

sh 'docker run --rm -v $(pwd):/workspace -w /workspace custom-python-snyk-image sh -c "ls -la PythonPlantsVsZombies"'

} else {

echo "PythonPlantsVsZombies directory does not exist inside the Docker container."

}

}

}

}

}

stage('Create Branch') {

steps {

sh '''

if git rev-parse --verify feature/update-date >/dev/null 2>&1; then

git checkout feature/update-date

else

git checkout -b feature/update-date

fi

'''

}

}

stage('Run Snyk Test First Time') {

steps {

script {

def osType = sh(script: 'uname', returnStdout: true).trim()

if (osType == "Darwin"){

// Capture the start time

def startTime = System.currentTimeMillis()

sh '''

cd PythonPlantsVsZombies

pwd

ls

python3 -m venv myenv

source myenv/bin/activate

pip install -r Requirements.txt

snyk auth ${SNYK\_TOKEN}

snyk test --package-manager=pip --file=Requirements.txt > snyk\_test\_output.txt

cat snyk\_test\_output.txt

exit 0

"

'''

// Calculate the duration in seconds

def duration = (System.currentTimeMillis() - startTime) / 1000

echo "First Snyk scan took ${duration} seconds"

env.SNYK\_SCAN\_DURATION= "${duration}"

}else{

// Capture the start time

def startTime = System.currentTimeMillis()

sh '''

docker run --rm -v $(pwd):/workspace -w /workspace -e SNYK\_TOKEN=${SNYK\_TOKEN} custom-python-snyk-image sh -c "

cd PythonPlantsVsZombies

python3 -m venv myenv

source myenv/bin/activate

pip install -r Requirements.txt

snyk auth ${SNYK\_TOKEN}

snyk test --package-manager=pip --file=Requirements.txt > snyk\_test\_output.txt

cat snyk\_test\_output.txt

exit 0

"

'''

// Calculate the duration in seconds

def duration = (System.currentTimeMillis() - startTime) / 1000

echo "First Snyk scan took ${duration} seconds"

env.SNYK\_SCAN\_DURATION= "${duration}"

}

}

}

}

stage('Display Snyk Output') {

steps {

sh '''

ls -la

[ -f PythonPlantsVsZombies/snyk\_test\_output.txt ] || touch PythonPlantsVsZombies/snyk\_test\_output.txt

cat PythonPlantsVsZombies/snyk\_test\_output.txt

'''

}

}

//This stage obtains the metrics and pushes them to pushgateway

stage('Push Results to Pushgateway') {

steps {

script {

// Initialize a global counter for metric pushes

def globalCounter = 0

// Check if there are any vulnerabilities detected

def vulnerabilitiesDetected = sh(script: """

grep -qE '\\[(Low|Medium|High|Critical) Severity\\]' PythonPlantsVsZombies/snyk\_test\_output.txt

""", returnStatus: true) == 0

// Initialize counters for each severity

def totalVulnerabilities = ['Low': 0, 'Medium': 0, 'High': 0, 'Critical': 0]

if (!vulnerabilitiesDetected) {

echo "No vulnerabilities detected."

} else {

// Parse the snyk\_output.txt file to extract severity ratings

def ratings = sh(script: """

grep -oE '\\[(Low|Medium|High|Critical) Severity\\]' PythonPlantsVsZombies/snyk\_test\_output.txt | cut -d'[' -f2 | cut -d' ' -f1 | sort | uniq -c

""", returnStdout: true).trim()

// Update the counters based on the parsed ratings

ratings.split("\n").each { line ->

def (count, rating) = line.split()

totalVulnerabilities[rating] += count.toInteger()

}

def issuesCounter = 0;

// Extract the names of all the issues associated with their severity level and push them to Pushgateway

['Low', 'Medium', 'High', 'Critical'].each { severity ->

def issues = sh(script: """

grep -oE "✗ [^\\\\[]+ \\\\[${severity} Severity\\\\]" PythonPlantsVsZombies/snyk\_test\_output.txt | awk -F '✗ ' '{print \$2}' | awk -F ' \\\\[${severity}' '{print \$1}' | tr '\n' ','

""", returnStdout: true).trim()

if (issues) {

echo "Pushing ${severity} vulnerabilities: ${issues}"

sh """

echo "snyk\_issues\_${issuesCounter}{severity=\\"${severity}\\", issues=\\"${issues}\\", instance=\\"${env.NODE\_NAME}\\"} 1" | curl --data-binary @- http://${JENKINS\_HOST}:9091/metrics/job/snyk\_scan

"""

issuesCounter++

}

}

}

// Push individual and total counts for each severity to Pushgateway

totalVulnerabilities.each { rating, count ->

echo "Pushing ${rating} vulnerabilities count: ${count}"

sh """

echo "snyk\_vulnerabilities\_${globalCounter}{severity=\\"${rating}\\", instance=\\"${env.NODE\_NAME}\\"} ${count}" | curl --data-binary @- http://${JENKINS\_HOST}:9091/metrics/job/snyk\_scan

"""

globalCounter++

}

// Define the weight mapping for each rating

def weightMap = [

'Low': 1,

'Medium': 2,

'High': 5,

'Critical': 10

]

def totalWeightedScore = 0

// Calculate the weighted score for each severity

totalVulnerabilities.each { rating, count ->

// Calculate the weighted score

def weightedScore = weightMap[rating] \* count

// Add the weighted score to the total

totalWeightedScore += weightedScore

}

// Echo and push the total weighted score

echo "Pushing total weighted score for vulnerabilities: ${totalWeightedScore}"

sh """

echo "snyk\_total\_vulnerabilities\_weighted{instance=\\"${env.NODE\_NAME}\\"} ${totalWeightedScore}" | curl --data-binary @- http://${JENKINS\_HOST}:9091/metrics/job/snyk\_scan

"""

def syncScanDurationCounter = 0;

// Push the Snyk scan duration to Pushgateway

echo "Pushing Snyk scan duration: ${env.SNYK\_SCAN\_DURATION} seconds"

sh """

echo "snyk\_scan\_duration\_seconds\_${syncScanDurationCounter}{instance=\\"${env.NODE\_NAME}\\"} ${env.SNYK\_SCAN\_DURATION}" | curl --data-binary @- http://${JENKINS\_HOST}:9091/metrics/job/snyk\_scan

"""

syncScanDurationCounter++

}

}

}

stage('Send Deployment Frequency Commits to Python Plants Vs Zombies') {

steps {

script {

// Count the number of commits affecting the PythonPlantsVsZombies directory across all branches in the past day

def commitCountInDir = sh(script: 'git log --oneline --all --since="1 day ago" -- PythonPlantsVsZombies/ | wc -l', returnStdout: true).trim()

// Send this commit count to gateway

sh """

echo 'git\_commit\_frequency\_daily\_PythonPlantsVsZombies ${commitCountInDir}' | curl --data-binary @- http://${JENKINS\_HOST}:9091/metrics/job/git\_metrics

"""

}

}

}

// This will fail when the fake issues are generated

stage('Fail if vulnerable') {

steps {

script {

// Check if there are any vulnerabilities detected

def vulnerabilitiesDetected = sh(script: """

grep -qE '\\[(Low|Medium|High|Critical) Severity\\]' PythonPlantsVsZombies/snyk\_test\_output.txt

""", returnStatus: true) == 0

if (vulnerabilitiesDetected) {

echo "failed due to vulnerabilities being detected."

exit 1

}

}

}

}

stage('Copy to Production / Merge') {

steps {

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

sh '''

git config credential.helper 'store --file=.git/credentials'

echo "https://${GIT\_USERNAME}:${GIT\_PASSWORD}@github.cs.adelaide.edu.au" > .git/credentials

# Checkout the feature branch

git checkout feature/update-date

if [ ! -d "production" ]; then

mkdir production

fi

if [ -d "PythonPlantsVsZombies" ]; then

# Check if package.json or package-lock.json exists inside PythonPlantsVsZombies and remove them

if [ -f "PythonPlantsVsZombies/package.json" ]; then

echo "package.json detected inside PythonPlantsVsZombies. Removing it before copying to production."

rm PythonPlantsVsZombies/package.json

fi

if [ -f "PythonPlantsVsZombies/package-lock.json" ]; then

echo "package-lock.json detected inside PythonPlantsVsZombies. Removing it before copying to production."

rm PythonPlantsVsZombies/package-lock.json

fi

cp -r PythonPlantsVsZombies/\* production/

# Modify a text document with the current time

echo "last commit: $(date)" > production/last\_commit.txt

git add production/

git commit -m "Copy PythonPlantsVsZombies to production directory and update last\_commit.txt"

# Checkout main and merge the feature branch

git checkout main

git pull origin main

git merge feature/update-date --no-ff --strategy-option theirs -m "Merge feature/update-date into main"

git push origin main

else

echo "Error: PythonPlantsVsZombies directory not found!"

exit 1

fi

'''

}

}

}

stage('Send Deployment Frequency Merges') {

steps {

script {

// Count the number of merges to the main branch

def mergeCount = sh(script: 'git log --oneline --merges --since="1 day ago" origin/main | wc -l', returnStdout: true).trim()

// Send this count to gateway

sh """

echo 'git\_deployment\_frequency\_daily ${mergeCount}' | curl --data-binary @- http://${JENKINS\_HOST}:9091/metrics/job/git\_metrics

"""

}

}

}

}

}

# A.17 Useful Links

The following sections include useful links that can be used to supplement the information provided in this document.

## A.17.1 Windows Based Agent Links

<https://www.youtube.com/watch?v=N8AQTlHoBKc>

<https://www.youtube.com/watch?v=VwliiaOI5po>

## A.17.2 EC2 Security Rules

[EC2 Instance Connect Tutorial](https://www.youtube.com/watch?v=lxSNeF7BAII&t=91s)

## A.17.3 Prometheus

Tutorial for install: <https://towardsaws.com/getting-started-with-prometheus-part-2-installation-of-prometheus-on-ec2-instance-b0fcb6201bb3>

<https://gist.github.com/darinpope/1c8422fb7512411760ccb2827d82613f>

## A.17.4 Grafana

<https://grafana.com/docs/grafana/latest/setup-grafana/installation/>

## A.17.5 Snyk and SAST

[CI CD Jenkins - Philippe Stemberger](https://www.youtube.com/watch?v=PHobD71mhYM)

[https://snyk.co/Sign-Up-YT](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbVprTDRTRFpFYU94YTVCb1ZJWFpLSm12M0FFUXxBQ3Jtc0trTl9yRGFQT1YzQjI3NFUzMVJ0ZEJPZW5RQnI1MVR2UUZzM3VOTnhmNzJOQmk2ZVpPMWRCRzZ0VEl2UGNXS1Bpcktod0U0UTZsNTAzSXVqSzQ5THBOdzVLcVU0XzBYS1kyMlB5enlNbWpTa08xTV9zQQ&q=https%3A%2F%2Fsnyk.co%2FSign-Up-YT&v=PHobD71mhYM)

<https://www.benq.com/en-au/business/resource/trends/what-is-cve-and-cvss.html>

# A.18 Plugin

To set up the Jenkins plugin the first step is to get Jenkins running on localhost.

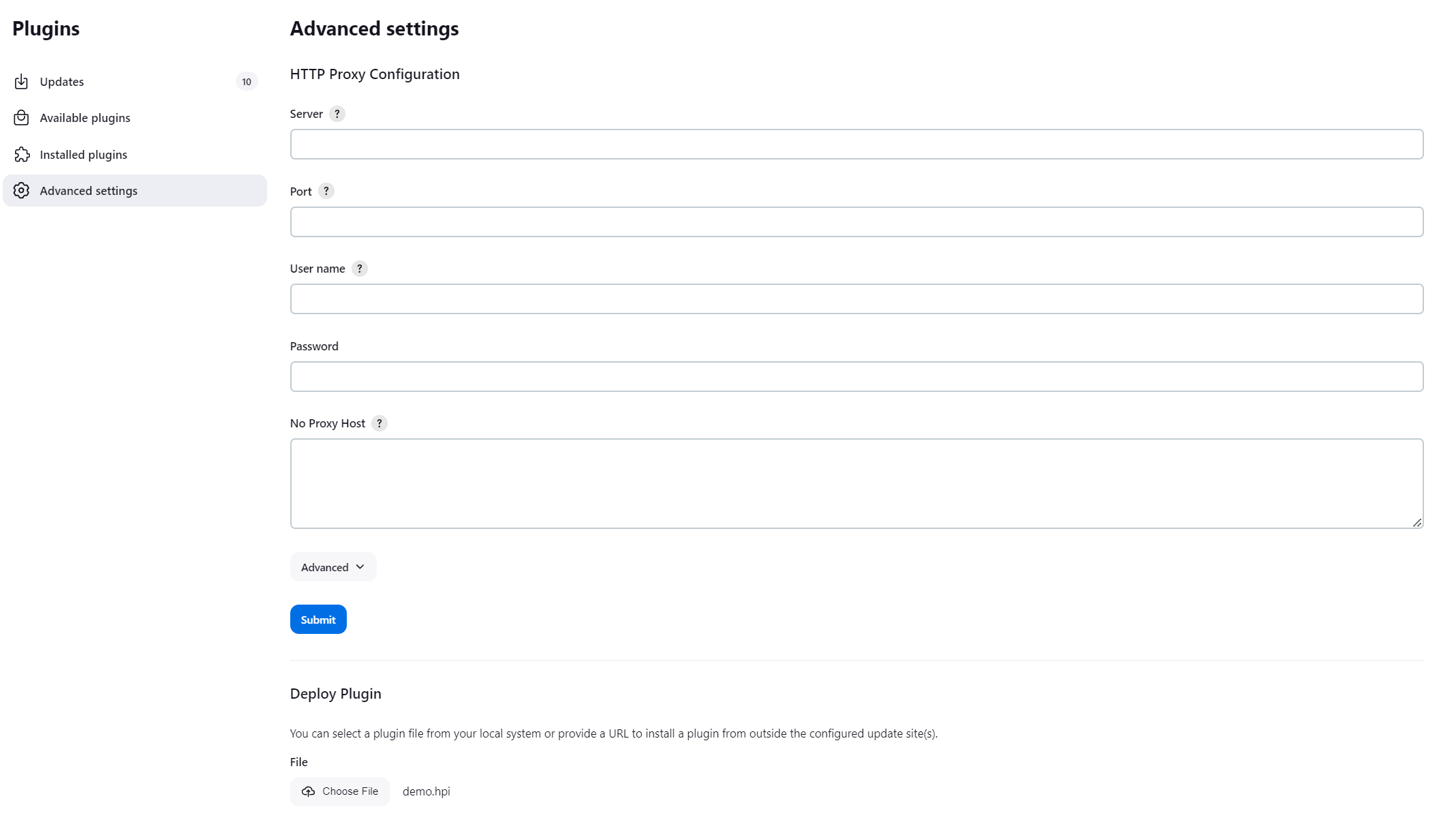
Then install and set up maven.

In the demo plugin folder of the repository run:

mvn clean package -DskipTests

This will generate the target file

Then you will need to add the hpi file through the advanced plugin menu



Then click deploy.

Then restart the Jenkins instance.

Then create a new freestyle project

And add new build step: say hello world (current name of build step inherited from demo, will be modified)

We still need to configure the credentials properly at this step but this is when you would select them after fully configured

Then click save.

Running this freestyle project will generate a pipeline which runs the jenkinsfile created for this plugin.

------- main files of note -------------------------------------------------------------------------

Path\To\Project\DORAAndSASTMetricJenkinsPlugin\CD1-main\JenkinsPlugin\demo-plugin\src\main\java\io\jenkins\plugins\sample\HelloWorldBuilder.java

Contains the java code for the plugin

Path\To\Project\DORAAndSASTMetricJenkinsPlugin\CD1-main\JenkinsPlugin\demo-plugin\src\main\resources\jenkinsfile

Is the Jenkinsfile we are loading with all the relevant Jenkinsfile code

Path\To\Project\DORAAndSASTMetricJenkinsPlugin\CD1-main\JenkinsPlugin\demo-plugin\src\main\resources\io\jenkins\plugins\sample\HelloWorldBuilder\config.jelly

Defines the drop downs which connect to our java code

------ please add to files of note as you understand how the plugin works more, file structure understanding not yet complete -----------------------------------------------------------------------------------

----- The following is for creating a plug in, you do not need to do this -------

Plugin creation:

https://www.jenkins.io/doc/developer/tutorial/create/

https://www.jenkins.io/doc/developer/tutorial/run/

Setup for current plugin

Step1: create and goto JenkinsPlugin folder

Step2:

mvn -U archetype:generate -Dfilter="io.jenkins.archetypes:"

…

Choose archetype:

1: remote -> io.jenkins.archetypes:empty-plugin (Skeleton of a Jenkins plugin with a POM and an empty source tree.)

2: remote -> io.jenkins.archetypes:global-configuration-plugin (Skeleton of a Jenkins plugin with a POM and an example piece of global configuration.)

3: remote -> io.jenkins.archetypes:global-shared-library (Uses the Jenkins Pipeline Unit mock library to test the usage of a Global Shared Library)

4: remote -> io.jenkins.archetypes:hello-world-plugin (Skeleton of a Jenkins plugin with a POM and an example build step.)

5: remote -> io.jenkins.archetypes:scripted-pipeline (Uses the Jenkins Pipeline Unit mock library to test the logic inside a Pipeline script.)

Choose a number or apply filter (format: [groupId:]artifactId, case sensitive contains): : 4

Choose io.jenkins.archetypes:hello-world-plugin version:

1: 1.1

2: 1.2

3: 1.3

4: 1.4

5: 1.5

6: 1.6

7: 1.7

8: 1.8

9: 1.9

10: 1.10

11: 1.11

12: 1.12

13: 1.13

14: 1.14

15: 1.15

16: 1.16

17: 1.17

18: 1.18

19: 1.19

20: 1.20

21: 1.21

Choose a number: 21: 21

…

[INFO] Using property: groupId = unused

[INFO] Using property: package = io.jenkins.plugins.sample

[INFO] Using property: hostOnJenkinsGitHub = true

Define value for property 'artifactId': demo

Define value for property 'version' 1.0-SNAPSHOT: :

Confirm properties configuration:

groupId: unused

package: io.jenkins.plugins.sample

hostOnJenkinsGitHub: true

artifactId: MetricPlugin

version: 1.0-SNAPSHOT

Y: : y

Step3:

mv MetricPlugin MetricPlugin-plugin

cd MetricPlugin-plugin

mvn verify