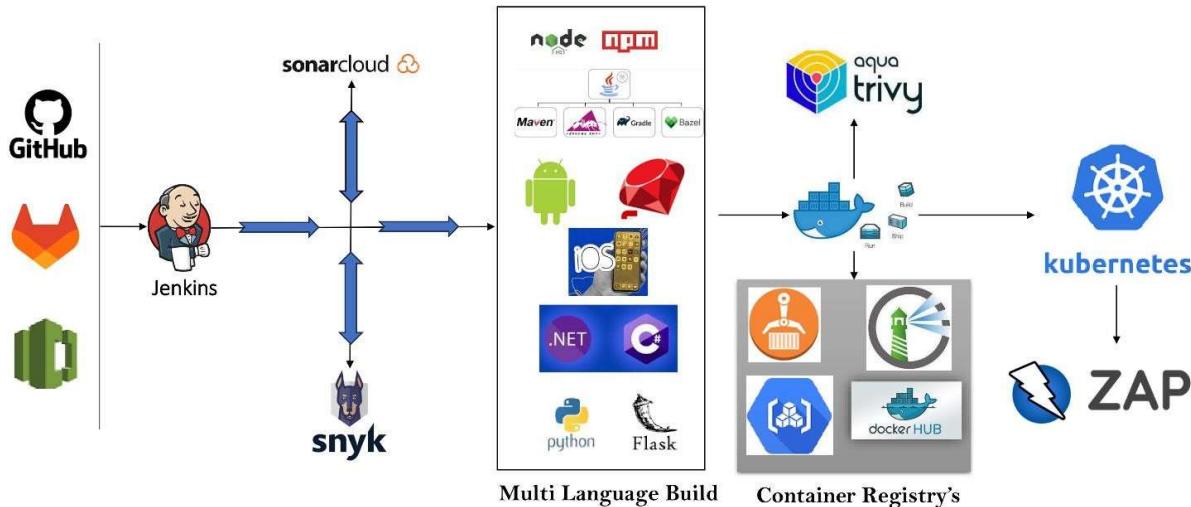


## DEVSECOPS PIPELINE: AUTOMATING CI/CD PIPELINE FOR SECURE MULTI-LANGUAGE APPLICATIONS USING JENKINS



### Prerequisites:

1. Git
2. Jenkins
3. Sonar-Scanner
4. Snyk
5. Application - Python, Java, Maven, Node etc... (The language you choose for your project depends on which one needs to be installed.)
6. Docker
7. Aqua Trivy
8. Kubernetes
9. Zaproxy

**Gitlab:** <https://gitlab.com/ganesharavind124/AnaCart>

### Introduction:

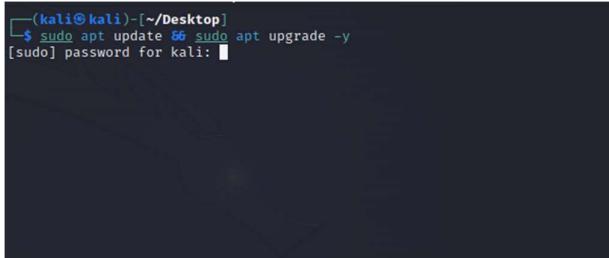
In today's fast-paced software development landscape, implementing efficient CI/CD pipelines is essential. This blog outlines the journey of building a robust CI/CD pipeline using Jenkins, integrating various tools to achieve seamless automation, security, and deployment for multi-language applications.

### Setting the Stage:

This project involved orchestrating a CI/CD pipeline that encompassed Git, SonarCloud, Synk, multi-language build automation, Docker, Aqua Trivy, Kubernetes, and ZAP Proxy. Leveraging Jenkins' flexibility and Groovy scripting capabilities, these tools are streamlined into a cohesive pipeline.

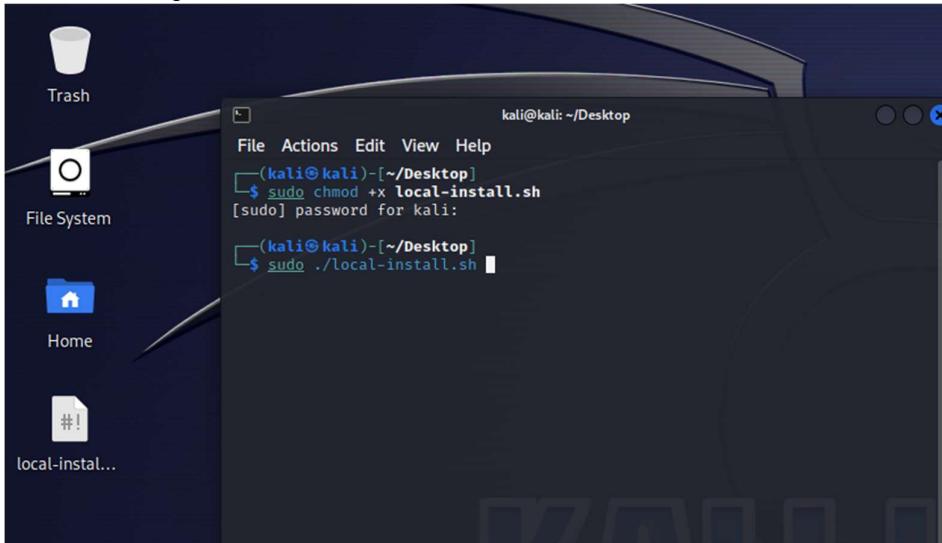
## Local Linux Installation:

1. Install and update Linux machine. For this demonstration, I am using Kali Linux

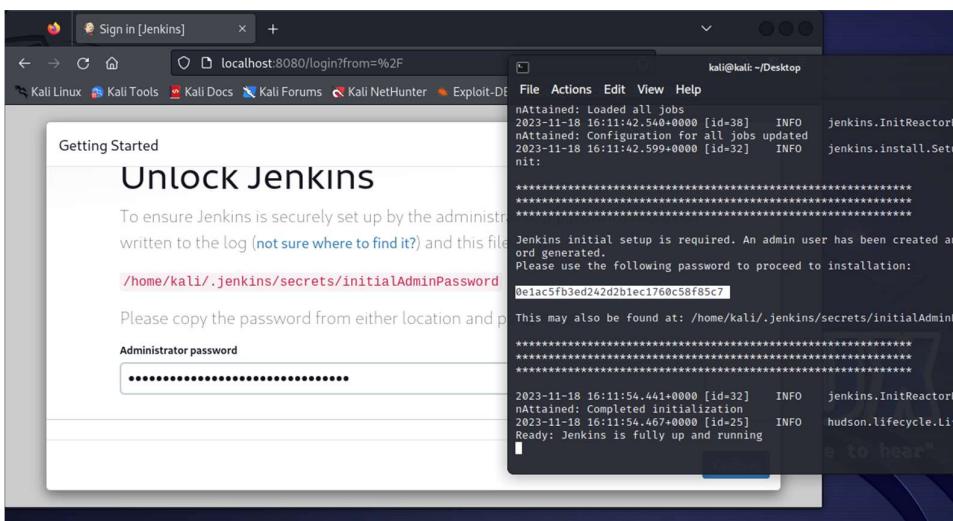


```
(kali㉿kali)-[~/Desktop]
$ sudo apt update && sudo apt upgrade -y
[sudo] password for kali: [REDACTED]
```

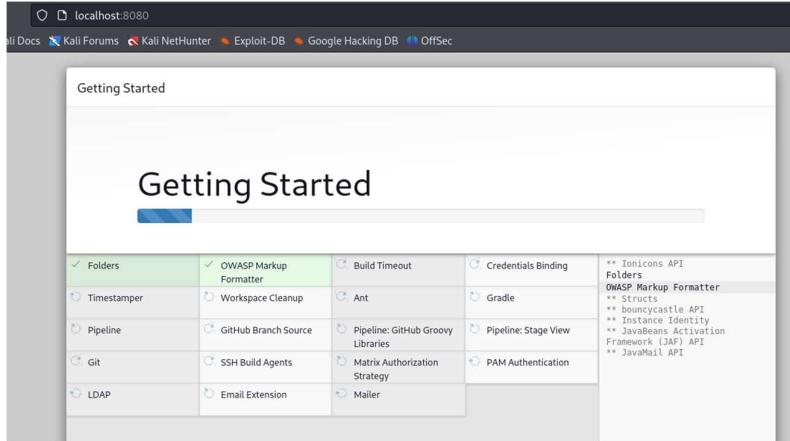
2. Download local-install.sh to Linux machine and place on Desktop
3. Make script executable
4. Run Install script



5. Run command ‘jenkins’ to start Jenkins and retrieve initial password
6. Log in to Jenkins with password that appears on screen by visiting <http://localhost:8080>
  - a. Keep this window open! The script simply executes the command Jenkins at the end which starts the Jenkins server process
  - b. If password does not appear or you close window by mistake just run cat for the location given after running the the ‘jenkins’ command
  - c. Paste password into ‘Administrator Password’ field



7. Click install suggested plugins and wait for installations to complete:



8. Create admin user credentials or use defaults for testing

### Installing Plugins for the pipeline

1. From the dashboard go to Manage Jenkins> Plugins> Available Plugins
2. Select SonarQube Scanner and Sonar Quality Gates for installation

Name	Description	Version	Last Updated
SonarQube Scanner	This plugin allows an easy integration of SonarQube, the open source platform for Continuous Inspection of code quality.	2.16.1	1 mo 9 days ago
Sonar Quality Gates	Fails the build whenever the Quality Gates criteria in the Sonar 5.6+ analysis aren't met (the project Quality Gates status is different than "Passed")	1.3.1	5 yr 3 mo ago

3. Select Snyk Security for installation

Name	Description	Version	Last Updated
SonarQube Scanner	This plugin allows an easy integration of SonarQube, the open source platform for Continuous Inspection of code quality.	2.16.1	1 mo 9 days ago
Sonar Quality Gates	Fails the build whenever the Quality Gates criteria in the Sonar 5.6+ analysis aren't met (the project Quality Gates status is different than "Passed")	1.3.1	5 yr 3 mo ago
Snyk Security	Add the ability to test your code dependencies for vulnerabilities against Snyk database	4.0.2	2 mo 8 days ago

4. Select Docker, Docker commons, Docker Pipeline, Docker API and docker-build-step for installation

The screenshot shows a list of Jenkins plugins selected for installation. Each item includes a checked checkbox, the plugin name, its version, a brief description, and the date it was last updated.

- Docker 1.5**: Version 439.va\_3cb\_0a\_6a\_fb\_29, last updated 2 mo 15 days ago. Description: This plugin integrates Jenkins with Docker.
- Docker Commons**: Version 439.va\_3cb\_0a\_6a\_fb\_29, last updated 4 mo 10 days ago. Description: Provides the common shared functionality for various Docker-related plugins.
- Docker Pipeline**: Version 572.v950f58993843, last updated 3 mo 9 days ago. Description: Build and use Docker containers from pipelines.
- Docker API**: Version 3.3.1-79.v20b\_53427e041, last updated 5 mo 16 days ago. Description: This plugin provides docker-Java API for other plugins. A note below says: "This plugin is up for adoption! We are looking for new maintainers. Visit our [Adopt a Plugin](#) initiative for more information."
- docker-build-step**: Version 2.10, last updated 1 mo 15 days ago. Description: This plugin allows to add various docker commands to your job as build steps.

5. Ensure that all of the selected Sonar, Docker, and Snyk dependencies are installed (Success) before moving on to next step

SonarQube Scanner	
Oracle Java SE Development Kit Installer	
SSH server	
Sonar Quality Gates	
Snyk Security	
Cloud Statistics	
Authentication Tokens API	
Docker Commons	
Apache HttpComponents Client 5.x API	<span style="width: 20%;">Installing</span>
Docker API	
Docker	
Docker Commons	
Docker Pipeline	
Docker API	
Javadoc	
JSch dependency	
Maven Integration	
docker-build-step	
Loading plugin extensions	

6. Install Kubernetes-cd manually for Kubernetes (this is missing from Jenkins plugin list)

a. You can use the plugin file from the repo at <https://github.com/Martian1337/BugFlow>

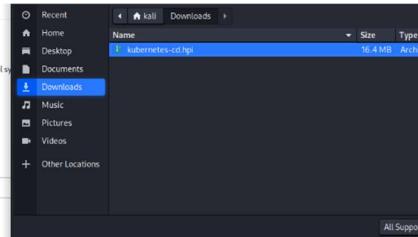
OR

b. You can also Download from <https://updates.jenkins.io/download/plugins/kubernetes-cd/1.0.0/kubernetes-cd.hpi>

The screenshot shows the Jenkins plugin index for the 'kubernetes-cd' plugin. It displays a table with one row, showing the plugin's name and details.

<b>Index of /plugins/kubernetes-cd/1.0.0</b>		
<a href="#">Parent Directory</a>	<a href="#">Last modified</a>	<a href="#">Size</a>
<a href="#">kubernetes-cd.hpi</a>	2020-08-20 12:33	16M

- c.  
d. Go to Manage Jenkins>Plugins>advanced settings  
e. Navigate to Deploy plugin section  
f. Click browse and select downloaded Kubernetes-cd .hpi file  
g. Click Deploy



h. Confirm plugin installed successfully

Docker Pipeline		Success
Docker API		Success
Javadoc		Success
JSch dependency		Success
Maven Integration		Success
docker-build-step		Success
Loading plugin extensions		Success
Azure Commons		Success
Preparation		
kubernetes-cd		Success

7. Install Kubernetes plugins – Kubernetes, Kubernetes Credentials, Client API, Kubernetes CLI

Plugin	Description	Last Updated
Kubernetes	This plugin integrates Jenkins with Kubernetes.	2 days 1 hr ago
Kubernetes Credentials	Common classes for Kubernetes credentials	2 mo 21 days ago
Kubernetes Client API	Kubernetes Client API plugin for use by other Jenkins plugins.	2 mo 21 days ago
Kubernetes CLI	Configure kubectl for Kubernetes	2 mo 20 days ago
Kubernetes Credentials Provider	Provides a read only credentials store backed by Kubernetes.	1 mo 1 day ago

8. Ensure Kubernetes plugins are installed successfully

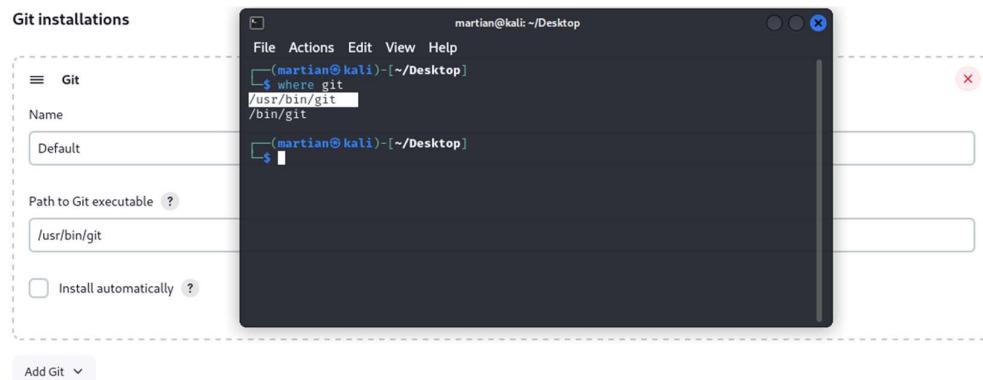
## Configure Jenkins Tools

1. Navigate to Manage>Tools to configure the tools installed via the install script
2. Configure the name and path for:

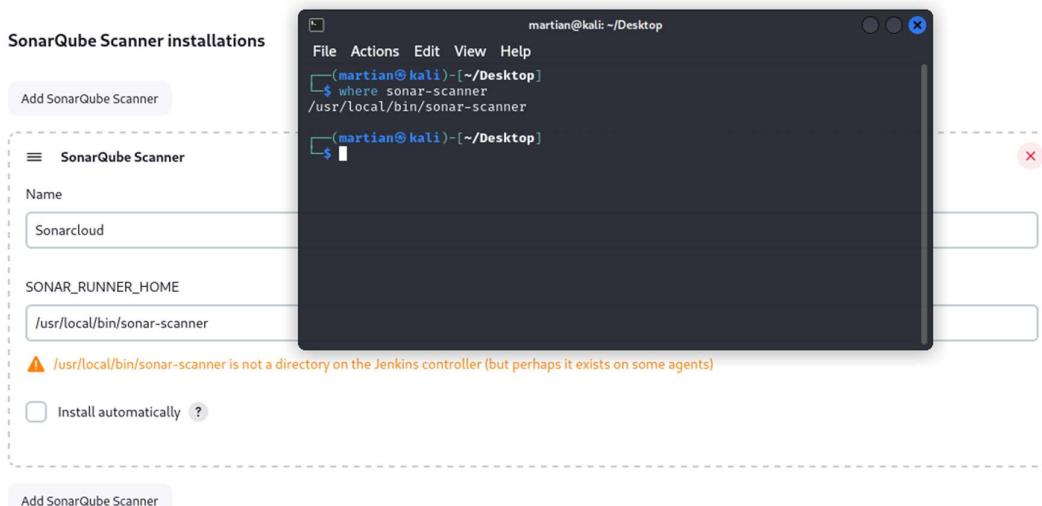
### a. JDK



### b. Git



### c. SonarQube scanner (sonar-scanner)



d. Snyk



e. Maven



f. Docker



Note: Versions may vary so ensure that the tool and environment variable match accordingly

Clone/fork the target repo

1. Go to the repo with the source code to be tested
2. Clone/fork the repo into your own Github/Gitlab

- Add/configure a deployment.yaml file for the docker instance to be deployed

Note: For this demo I am using an app from the OWASP Vulnerable Applications Directory

### Configure Credentials

- Go to Manage Jenkins>Credentials>Click Global under domains for System store
- Click Add credentials

ID	Name	Kind	Description
			This credential domain is empty. How about <a href="#">adding some credentials?</a>

Icon: S M L

- Add Snyk credentials

- Login to snyk account <https://snyk.io> and click skip for now
- Go to Snyk account settings and generate auth token
- Copy auth token

**Auth Token**  
Use this token to authenticate the Snyk CLI and in CI

KEY

Scope ?  
Global (Jenkins, nodes, items, all child items, etc)

Token ?

ID ?  
Snyk

Description ?

**Save**

4. Add Sonarcloud credentials
  - a. Login to Sonarcloud and start a new project
  - b. Copy generated token
  - c. Select ‘Secret text’ from dropdown menu
  - d. Paste Sonarcloud token into ‘Secret field
  - e. Name this token Sonarcloud

Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted) >

### New credentials

Kind

Secret text

Scope ?

Global (Jenkins, nodes, items, all child items, etc)

Secret

\*\*\*\*\*

ID ?

Sonarcloud

Description ?

Create

- f. Add Sonarcloud URL as Server to system by visiting **Manage Jenkins> System> SonarQube Servers**

If checked, job administrators will be able to inject a SonarQube server configuration as environment variables in the build.

Environment variables

SonarQube installations

List of SonarQube installations

Name	X
Sonarcloud	
Server URL	Default is http://localhost:9000
https://sonarcloud.io	
Server authentication token	SonarQube authentication token. Mandatory when anonymous access is disabled.
Sonarcloud	
+ Add ▾	

## 5. Add credentials for Docker

- a. Copy Docker username/password used for docker account
  - b. In Jenkins, user/pass credentials into appropriate fields
  - c. Name the credentials “Docker\_Server” in the ID field as written in the pipeline script
  - d. Click Create

Kind Username with password

Scope Global (Jenkins, nodes, items, all child items, etc)

Username Martian 1337

Treat username as secret

Password

ID Docker\_Server

## Add Kubernetes kubeconfig to Jenkins

- e. Start minikube with ‘`sudo usermod -aG docker $USER && newgrp docker && minikube start`’ so that the config file can be retrieved
  - f. Run command ‘`cat ~/.kube/config > kubeconfig.txt`’ from the Linux Desktop
  - g. Open kubeconfig.txt with text editor
  - h. Display the certificate in the path in the ‘certificate-authority’ field and turn the certificate into base64.

You can do this in one line by typing as ‘`cat /path/to/certificate | base64 -w 0; echo`’ shown here:

```
File Edit Search View Document Help
File Actions Edit View Help
(kali㉿kali: ~/Desktop)
└─$ ls /home/kali/.minikube/ca.crt
/home/kali/.minikube/ca.crt  basedir  cert  key  echo
1 apiVersion: v1
2 clusters:
3 - cluster:
4   certificate-authority: /home/kali/.minikube/ca.crt
5   extensions:
6     - extension:
7       last-update: Sat, 18 Nov 2023 16:07:00 EST
8       provider: minikube.sigs.k8s.io
9       version: v1.32.0
10      name: cluster.info
11    server: https://192.168.49.2:8443
12    name: minikube
13 contexts:
14 - context:
15   cluster: minikube
16   extensions:
17     - extension:
18       last-update: Sat, 18 Nov 2023 16:07:00 EST
19       provider: minikube.sigs.k8s.io
20       version: v1.32.0
21     name: context_info
22   namespaceDefault
23   name: minikube
24   name: minikube
25 current-context: minikube
26 kind: Config
27 preferences: {}
28 users:
29 - name: minikube
30   user:
31     client-certificate: /home/kali/.minikube/profiles/minikube/client.crt
32     client-key: /home/kali/.minikube/profiles/minikube/client.key
33
```

- i. Paste the base64 encoded certificate into the ‘certificate-authority’ kubeconfig.txt as shown below

- j. Scroll down to 'client-certificate' and repeat that process for 'client.crt' information as shown below

- k. Repeat this process for ‘client-key’ for ‘client.key’ information as shown below

1. After pasting the base64 encoded certificate data, **add '-data'** to each field like below:

- m. Now go to Jenkins, create credential for ‘kubernetes’
  - n. Paste the new kubeconfig contents directly into the kubeconfig
  - o. add ‘-data’ to the client-authority, client-certificate, and client-key fields
  - p. Name it ‘kubernetes’ and click create

The final kube contents in Jenkins should look like the following:

#### Certificate-authority-data:

Kubeconfig

Enter directly

Content ?

```
apiVersion: v1
clusters:
- cluster:
  certificate-authority-data:
LS0tLS1CRUdjTIBDRVJUSUZJQ0FURS0tLS0tCk1JSURCAkNDQWU2Z0F3SUJBZ0lCQVRBTkJna3Foa2lHOXcwQkFRc0ZBREFWTvJnd0RWURWUVFERXdwGFXNXAKYTNWaVpVTkJNQjRYRFRJeKtURxhOekl4TURZME4xb1hEVE16TVRFcE5USxhNRfkWtFvd0ZURVRNQkVHQTFRQpBeE1LYldsDFzdVZEUVRDQ0FTSxdEUtKS29aSwh2Y05BUUVVCQlFBRGdnRBRENDQFvQ2dnRUJBs25yCnNpQ09sQ21lYzFrlzOFQvWlpYzVkeM5pSk9COXQ1MHpWVkc3VVFVN2lQ3NZ5FhlTjQyR2tVgdmMUJvR3QdkdU4N2UmjQraTN2WUNZZEJUTY1dXlocGEQjZqckNDYzNaSDNjVzhSem9jOXtL1QzU3hWM1hdDdZVHIOOApjDU5dfVHNW9NdGFhaje0cvlyNnRa3nVWnNa4b1FKSXbZnBQZT2FRNkRlQWVmUk9mc2lRQVMwRVlrcitaQWY3CkVon3pBOUZqSiNwNjtsvRxclNhY2RjdONFZkTepvV3NkTrJNcVkrBzU2j5T2FqdE6jChF1UZDR1NHfWVDMkbkFnblczqjZdHXM2xpU1RWL2tVWkNjckSMUkixTUSCz2mTlBBCvYYXBleFozMOV3aUdqR0FEZH6NuNyTQpwWk1UWhpmgbGFOs2MrVvU2jRfFNQ0F3RUFByU5oTUy4dOrnWURWUJBQQVFL0jBUURBZ0tTUlwR0ExVWRKUVFXCk1CUUDQ3NHQVFVRk3TUNCZ2dyQmdFRkYURBVbEFQQmdOvkhsTUIJBjhFQlRBREFRSC9NjQhGHTFVZERnUvCkQkjSdDjsaOfmteVpUhlyRvdBcUsxejZEaTN6dUf6Q5CZ2xaGtPz13MEjBUXNGQUFPQRUFLSXVUSWRBcApRam11vn12YKnidGh0WUUpXcm9L52hrURhbWtaK0tDWGJpREvQkZyM2IMbkhbHVha3JRTXZSRUjsUzd0MWFWClwMW5jYi9PYmNo2hFRk96N0lWw3liYlo1UVBONFkwM2J4YUU4dFFXeWxRsmaJyBwS0ZvdFBJRw0wdFBISGwKQWZraVBPyNPNOTgrQ3jVcW4wQmFWUFC3ZwcwU3dTFqbTkboZxZdG00bCtGhdzVEFrcktYelR2tn2QzlZMapXyZfESjObxgzbWdpRFNytTxNRFhMMHdyQm95QjU0U05oOGRpcljVdlwQzjGvVFFWFZjSxpLctVnUVjkZlBHCmtyaBzQnASyNlEbmu4NEp2T3ljR3B2WlnTj2VoMDQz0Vh6bxLvdCswb01FSTlBbd4QXJ3dT1RG12MHjtTVEKMyt4TnM5c0FsTzdiaC9PQotLS0tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg=
```

extensions:

- extension:

#### Client-certificate-data and client-key-data:

Enter directly

Content ?

```
-----BEGIN CERTIFICATE-----  
user:  
  client-certificate-data:  
LS0tLS1CRUdjTIBDRVJUSUZJQ0FURS0tLS0tCk1JSURCAkNDQWU2Z0F3SUJBZ0lCQVRBTkJna3Foa2lHOXcwQkFRc0ZBREFWTvJnd0RWURWUVFERXdwGFXNXAKYTNWaVpVTkJNQjRYRFRJeKtURxhOekl4TURZME4xb1hEVE16TVRFcE5USxhNRfkWtFvd0ZURVRNQkVHQTFRQpBeE1LYldsDFzdVZEUVRDQ0FTSxdEUtKS29aSwh2Y05BUUVVCQlFBRGdnRBRENDQFvQ2dnRUJBs25yCnNpQ09sQ21lYzFrlzOFQvWlpYzVkeM5pSk9COXQ1MHpWVkc3VVFVN2lQ3NZ5FhlTjQyR2tVgdmMUJvR3QdkdU4N2UmjQraTN2WUNZZEJUTY1dXlocGEQjZqckNDYzNaSDNjVzhSem9jOXtL1QzU3hWM1hdDdZVHIOOApjDU5dfVHNW9NdGFhaje0cvlyNnRa3nVWnNa4b1FKSXbZnBQZT2FRNkRlQWVmUk9mc2lRQVMwRVlrcitaQWY3CkVon3pBOUZqSiNwNjtsvRxclNhY2RjdONFZkTepvV3NkTrJNcVkrBzU2j5T2FqdE6jChF1UZDR1NHfWVDMkbkFnblczqjZdHXM2xpU1RWL2tVWkNjckSMUkixTUSCz2mTlBBCvYYXBleFozMOV3aUdqR0FEZH6NuNyTQpwWk1UWhpmgbGFOs2MrVvU2jRfFNQ0F3RUFByU5oTUy4dOrnWURWUJBQQVFL0jBUURBZ0tTUlwR0ExVWRKUVFXCk1CUUDQ3NHQVFVRk3TUNCZ2dyQmdFRkYURBVbEFQQmdOvkhsTUIJBjhFQlRBREFRSC9NjQhGHTFVZERnUvCkQkjSdDjsaOfmteVpUhlyRvdBcUsxejZEaTN6dUf6Q5CZ2xaGtPz13MEjBUXNGQUFPQRUFLSXVUSWRBcApRam11vn12YKnidGh0WUUpXcm9L52hrURhbWtaK0tDWGJpREvQkZyM2IMbkhbHVha3JRTXZSRUjsUzd0MWFWClwMW5jYi9PYmNo2hFRk96N0lWw3liYlo1UVBONFkwM2J4YUU4dFFXeWxRsmaJyBwS0ZvdFBJRw0wdFBISGwKQWZraVBPyNPNOTgrQ3jVcW4wQmFWUFC3ZwcwU3dTFqbTkboZxZdG00bCtGhdzVEFrcktYelR2tn2QzlZMapXyZfESjObxgzbWdpRFNytTxNRFhMMHdyQm95QjU0U05oOGRpcljVdlwQzjGvVFFWFZjSxpLctVnUVjkZlBHCmtyaBzQnASyNlEbmu4NEp2T3ljR3B2WlnTj2VoMDQz0Vh6bxLvdCswb01FSTlBbd4QXJ3dT1RG12MHjtTVEKMyt4TnM5c0FsTzdiaC9PQotLS0tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg==  
  client-key-data:  
LS0tLS1CRUdjTIBSU0EgUFJJVkfURSBLRVktLS0tLQpNSUlfB3dJQkFBS0NBUVBbXvOeFZPRGNrZDNQUTU1aV1EaDFJMURhL1h5cmVYUjh1NFdPSXpvUlVkbTBhZFZMCmtmZVZeVZlbn4N1lkldUdTE4RVJld2kvOUdoNlh3M0Y0S3U1Z1XY055G3R653h3eWhnbm5HYmxhZHsOVQkdUrZvHuZb1hRFg3WEpQZ3NxVlFFVnFuOHFTQ3VMZU9wc0JuMFZdIa3S0xFazkyVCscnpaMLNMUXBq3ZjQgpkYXN4UExXYzNYTdrVisxUJxtSy9LbmJoc085ZEycXlbUROHAyc3lHUWhHMkt3cVaySURM3o1VEo2NOFDci4ZkZnZ5xvW54MWZlZG1ZTXRER3RteVvK0d3Y3jb3MsNhNEWHFK21PdmaNgxKREtdmExUghWVm94dTkkc1JuaU5neUtTdGkzUHvHyM1FEZkVsRmh6VjBjTm1URXF1QjV2V3dJREFRQUJbb0lCQUU1SG1PV081NnNjOFGZNQpZTHJFamQ4VlpEZ3NvL1UwUEQ0Qje3NNVnBS0k5ckLN6GZBTC9vQmFiMm5XeEg05nJ1NmFiWTlLM3FGQjQ2Y3BqCjNqNWhnUWnpYXRvNUjCGWRxcGFTK2FXWnfNSGw0TDl3S2oyNERkYjRjUvtrIRhpYUXZ3VktxMWE1Lop1vExWHgKN3pTemxVYUNZNhMcjZISGEwsSwry9RFJPWm5WVlhuaTzJvxdVMEpRS9CSHJdeUZBL0RKbGxTaH24RzNjMOpZeldieERTGkrVXBV23k1YzdnRE0zMGLOOUREeGxOVzhu0MzeWta1FORWwvFhltJfxZElwbEpQxVpHQUF3CjchBc05wMkdNFh1Tmh6TFB3Y0k4TGFxenp0dTZZM2F1cG14TjVWQuTPbUpmZjdYVJUJmpMQkluOghlNEVPd0wKeFU4MUZ3RUNnWUVBeg9MZFovNFVqVd1VeGvSeHzqU0NSM2RncUg50HRYNGlkMVFSZTYveE1DUE85aHp0SndXVgpMOFd1MklMRVVGRI1feUViQWloSXJVU22VdcDWTkvNVBDN0oOUxiQnrgQXRzWEFVMkY2V1ISUXZFZGpLdm5iClzMm3SVBqdWXRGp3SXZFNjBaalk0Y9IR1NnK3QwcUtFVXZkNWVvUJGNQmlhBMHUwTld5TUVDZ1fFQXg3NSsKZ1ZxWnpDYyt1ZWFreEFmZCtpRXN5cjMyTxE1VFZ0-----
```

6. Confirm that all credentials are added to match the picture below:

### Global credentials (unrestricted)

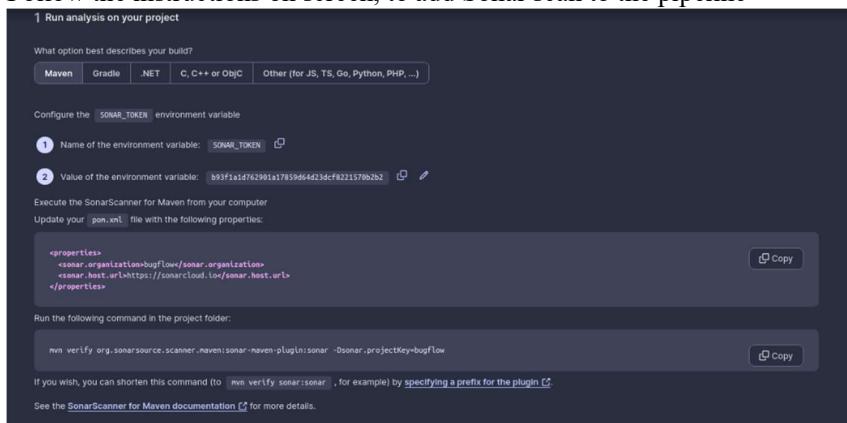
+ Add Credentials

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind	Description
Docker_Server	martian1337*****	Username with password	
Snyk	Snyk	Snyk API token	
kubernetes	kubernetes	Kubernetes configuration (kubeconfig)	
Sonarcloud	Sonarcloud	Secret text	

7. Add SonarCloud API key to project

- Log in to Sonarcloud and go to “My Account”
- In Sonarcloud, go to “My Projects” page and configure the project
- In the “Choose your analysis method” section, select your CI tool. For this application, Maven is being used.
- Follow the instructions on screen, to add Sonar scan to the pipeline



- e. Add the generated token to the pipeline script for continuity

```
stage('Compile and Run Sonar Analysis'){
    steps {
        script {
            withSonarQubeEnv(credentialsId: SONARCLOUD, installationName: 'Sonarcloud') {
                try {
                    if (fileExists('pom.xml')) {
                        sh "mvn verify org.sonarsource.scanner.maven:maven-scanner:sonar -Dsonar.projectKey=bugflow"
                    } else if (fileExists('package.json')) {
                        sh "$sonar.organization=jenkeen -Dsonar.projectKey=jenkeen_testjs -Dsonar.sources=. -Dsonar.host.url=https://sonarcloud.io -Dsonar.login=b93f1a1d762901a17859d64d23dcf8221570b2b2"
                    } else if (fileExists('go.mod')) {
                        sh "$sonar.organization=jenkeen -Dsonar.projectKey=jenkeen_go -Dsonar.sources=. -Dsonar.host.url=https://sonarcloud.io -Dsonar.login=b93f1a1d762901a17859d64d23dcf8221570b2b2"
                    } else if (fileExists('Gemfile')) {
                        sh "$sonar.organization=jenkeen -Dsonar.projectKey=jenkeen_ruby -Dsonar.sources=. -Dsonar.host.url=https://sonarcloud.io -Dsonar.login=b93f1a1d762901a17859d64d23dcf8221570b2b2"
                    } else if (fileExists('requirements.txt')) {
                        sh "$sonar.organization=jenkeen -Dsonar.projectKey=jenkeen_python -Dsonar.sources=. -Dsonar.host.url=https://sonarcloud.io -Dsonar.login=b93f1a1d762901a17859d64d23dcf8221570b2b2"
                    } else {
                        currentBuild.result = 'FAILURE'
                        pipelineError = true
                        error("Unsupported application type: No compatible build steps available.")
                    }
                } catch (Exception e) {
                    currentBuild.result = 'FAILURE'
                    pipelineError = true
                    error("Error during Sonar analysis: ${e.message}")
                }
            }
        }
    }
}
```

f. After updating pom.xml (Maven), it should now look like this in the repo when complete:

```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <easybuggy-port>8080</easybuggy-port>
    <!-- default properties (JDK 1.7 or earlier) -->
    <jvm.args.perm.size>-XX:MaxPermSize=128m</jvm.args.perm.size>
    <jvm.args.gc.log.path>-Xloggc:logs/gc.log</jvm.args.gc.log.path>
    <jvm.args.print.heap.at.gc>-XX:+PrintHeapAtGC</jvm.args.print.heap.at.gc>
    <jvm.args.print.gc.details>-XX:+PrintGCDetails</jvm.args.print.gc.details>
    <jvm.args.print.gc.date.stamps>-XX:+PrintGCDateStamps</jvm.args.print.gc.date.stamps>
    <jvm.args.gc.log.file.rotation>-XX:+UseGCLogFileRotation</jvm.args.gc.log.file.rotation>
    <jvm.args.number.of.gc.log.files>-XX:NumberOfGCLogFiles=5</jvm.args.number.of.gc.log.files>
    <jvm.args.gc.log.file.size>-XX:GCLogFileSize=10M</jvm.args.gc.log.file.size>
    <sonar.organization>bugflow</sonar.organization>
    <sonar.host.url>https://sonarcloud.io</sonar.host.url>
</properties>
```

## Local Kubernetes Deployment config

1. Run ‘`cat ~/kube/config`’ or open the created `kubeconfig.txt` to see the server URL
2. Paste the service URL into pipeline script’s Kubernetes Deployment stage

The screenshot shows a terminal window and a Jenkins Pipeline configuration interface.

In the terminal window, the command `cat ~/kube/config` is run, displaying the contents of the kubeconfig file. The file includes details about clusters, contexts, and a current context set to 'minikube'. It also shows a step in a Jenkins Pipeline script that uses the obtained URL.

In the Jenkins Pipeline configuration, a 'Pipeline' stage is defined. It contains a 'Script' block with the following Jenkins Pipeline code:

```
steps {
    script {
        try {
            def targetURL = "http://192.168.49.2:8443" // Use the obtained URL
            def zapCommand = "zapProxy -cmd -quickurl ${targetURL}"
            sh zapCommand
        }
    }
}
```

The Jenkins interface shows the pipeline stages and their execution status.

## Pipeline Config Methods:

Go to the configuration page in the pipeline job. This page will open. Add your Jenkins pipeline script path there. There are two options available.

1. Script for pipeline: Here is where you can write your own script.
2. Pipeline from SCM: it will use your SCM repository's Jenkins file.

Note: Method 2 requires that you have the Jenkins file in the same repo as the app.

Here is an example of the first option pasted directly pipeline:

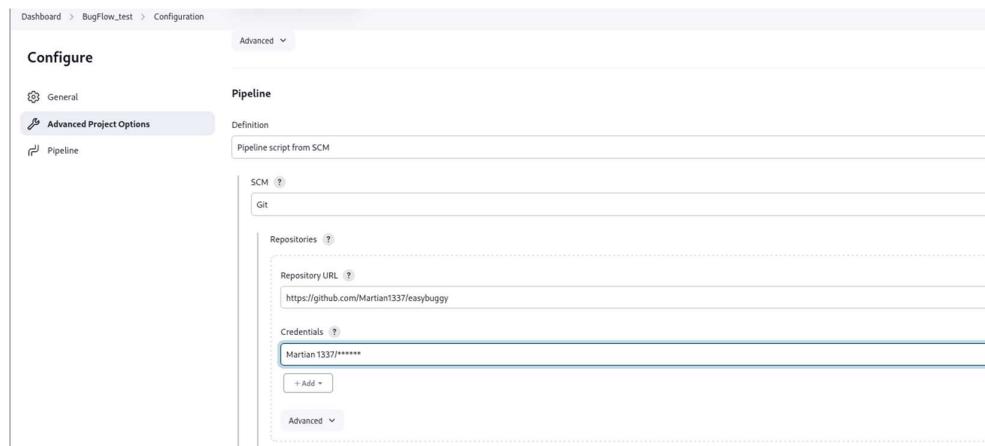


The screenshot shows the Jenkins 'Advanced Project Options' configuration page for a pipeline job. The 'Pipeline' section is selected. Under 'Definition', the 'Pipeline script' field contains the following Groovy code:

```
11-     }
12-     } else if (javaVersion.startsWith("11")) {
13-         return "11"
14-     } else if (javaVersion.startsWith("17")) {
15-         return "17"
16-     } else {
17-         error("Unsupported Java version detected: ${javaVersion}")
18-     } else {
19-         error("Java version information not found in output.")
20-     }
21- }
22- pipeline {
23-     agent any
24-     environment {
25-         SONARCLOUD = 'Sonarcloud'
26-         SNYK INSTALLATION = 'snvk@latest'
```

Below the code, there is a checked checkbox for 'Use Groovy Sandbox'. At the bottom, there are 'Save' and 'Apply' buttons.

Here is an example of the second option:



So, choose your SCM and give your branch and URL of your repository and also mention your Jenkinsfile in script path

Additional Behaviours

Script Path 

Lightweight checkout 

[Pipeline Syntax](#)

Create a pipeline job and give it a name of your choosing, such as BugFlow.

Jenkins Search (CTRL+K)

Dashboard >

Enter an item name

BugFlow\_test » Required field

**Freestyle project**  
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

**Maven project**  
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

**Pipeline**  
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

**Multi-configuration project**  
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

**Folder**  
Creating a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a namespace, so you can have multiple things of the same name as long as they are in different folders.

After clicking Ok, this is where the pipeline method can be selected; Script or SCM. Here I pasted the contents of the Jenkins file directly into the text box for “pipeline script”

Dashboard > BugFlow\_test > Configuration

**Configure**

General

Advanced Project Options

Pipeline

**Advanced Project Options**

**Advanced**

**Pipeline**

Definition

```

Script ?  

358 -      if (!fileExists(configFile)) {  

359 -          KubernetesDeployer(configFile, kubeconfigId: KUBE_CONFIG, namespace: namespace)  

360 -      } else {  

361 -          error("Error: configFile does not exist")  

362 -      }  

363 -      currentBuild.result = "FAILURE"  

364 -      pipelineError = true  

365 -  }  

366 -}  

367 -}  

368 -}  

369 -}
370 -}
371 -}
372 -}
373 -}

```

Use Groovy Sandbox 

[Pipeline Syntax](#)

Add URL of target repo to the pipeline script

The screenshot shows the Jenkins 'Configure' screen for a pipeline job. The 'Advanced Project Options' tab is selected. In the 'Pipeline' section, the 'Definition' is set to 'Pipeline script'. The script editor contains the following Groovy code:

```
30 DOCKER_TOOL = 'Docker' // Mention same name in Jenkins` in ID field
31 DOCKER_URL = 'https://index.docker.io/v1/' // Change not mandatory
32 KUBE_CONFIG = 'kubernetes'
33 }
34+
35 stages {
36   stage('Clean Workspace') {
37     steps {
38       cleanWs()
39     }
40   }
41   stage('Git Checkout') {
42     steps {
43       script{
44         git 'https://github.com/Martian1337/easybuggy'
45       }
46     }
47 }
48 }
```

A blue box highlights the line `git 'https://github.com/Martian1337/easybuggy'` in the script. Below the script editor, there is a checkbox labeled 'Use Groovy Sandbox' which is checked. At the bottom are 'Save' and 'Apply' buttons.

After creating the pipeline job will look like this:

The screenshot shows the Jenkins dashboard with the pipeline job 'BugFlow\_test' selected. The job card for 'BugFlow\_test' includes the following details:

- Status: Build Now
- Configure
- Delete Pipeline
- Full Stage View
- Rename
- Pipeline Syntax

The 'Stage View' section displays the message: "No data available. This Pipeline has not yet run."

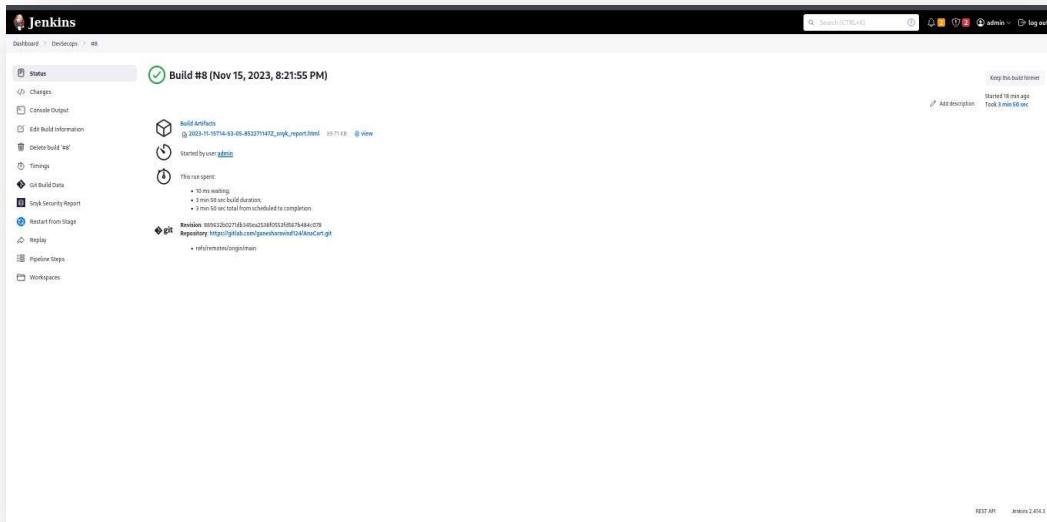
The 'Build History' section shows "No builds".

At the bottom, there are links for "Atom feed for all" and "Atom feed for failures".

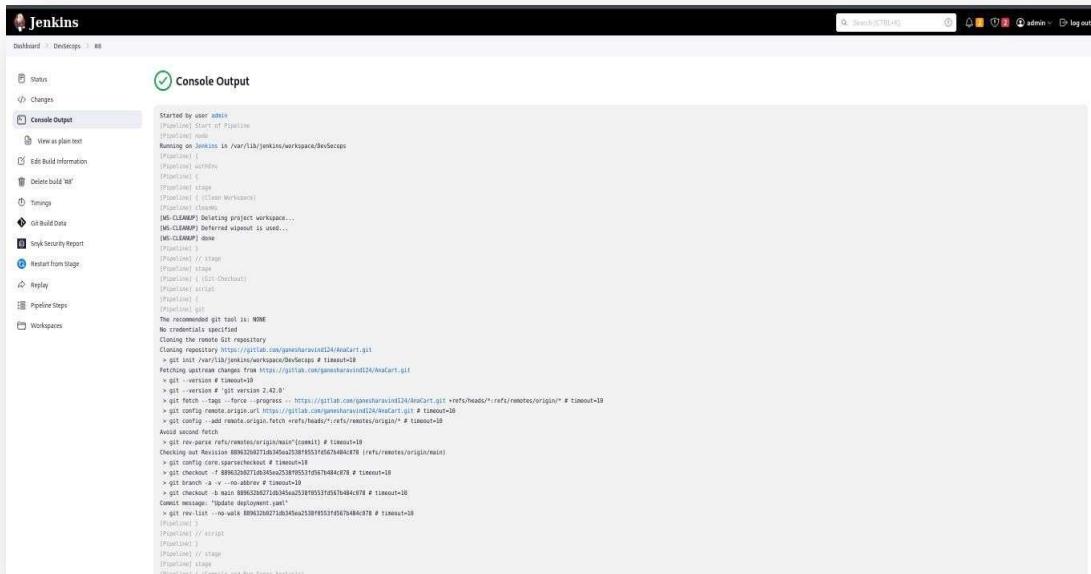
Check all of the lines, curl braces, and credentials before saving and applying. You should

also make sure that the variable names in the environment and the stages are the same, as many people make mistakes in this specific area. Next, click "Apply." If you run into any problems, an X will appear in that line. If you change "Save," the page will redirect to the main site.

After that, click the "Build Now" button.



The job will begin to execute. You can check your job outcomes at the console to see if there are any problems.



This is where you can see output of the job has succeeded.

The screenshot shows the Jenkins Pipeline DevSecOps dashboard. At the top, there's a navigation bar with links for Dashboard, Pipeline DevSecops, Status, Changes, Build Now, Configure, Delete Pipeline, Full Stage View, SonarQube, Rename, and Pipeline Syntax. On the right side of the header, there are links for Add description, Disable Project, and log out.

**Stage View**: This section displays a horizontal timeline of pipeline stages. The stages listed from left to right are: Clean Workspace, Git-Checkout, Compile and run Sonar Analysis, snyk\_analysis, Detect and set Java, Frontend Build and Test, Java Spring Boot Build and Test, .NET Build and Test, PHP Build and Test, iOS Build and Test, Android Build and Test, Ruby on Rails Build and Test, Flask Build and Test, Django Build and Test, Rust Build and Test, Ruby Sinatra Build and Test, Build Docker Image, Trivy Scan, Push Docker Image, Kubernetes Deployment, Run DAST Using ZAP, and Run DAST Using Zap. Each stage is represented by a colored bar indicating its duration. A tooltip for the first stage, "Clean Workspace", shows an average stage time of 70ms.

**SonarQube Quality Gate**: This section shows the status of the SonarQube quality gate. It indicates that the build has passed ("Passed") and that server-side processing is successful. It also lists the last four builds:

- Last build (#8), 7 min 1 sec ago
- Last stable build (#8), 7 min 1 sec ago
- Last successful build (#8), 7 min 1 sec ago
- Last completed build (#8), 7 min 1 sec ago

At the bottom right of the dashboard, there are links for REST API and Jenkins 2.414.3.

## STAGE 1 (Cleanup Workspace)

In this stage, we going to cleanup our workspace where files and documents that are previously deployed before, after this stage done the git will pull newly updated files and run everything newly.

```
stages {  
    stage('Clean Workspace') {  
        steps {  
            cleanWs()  
        }  
    }  
}
```

## Stage 2 (Git Checkout)

**Git:**



Give your Git details in the SCM above; therefore, update those with the URL and branch names of your git details in from SCM.

```
stage('Git-Checkout') {  
    steps {  
        checkout scm  
    }  
}
```

**Note:** If your git repository is private, you should give your Jenkins account your Gitlab personal access token or git credentials.

### Stage 3 (Sonarcloud):

SonarCloud is used to perform SAST code Quality scans, so integrate it with Jenkins by giving it a personal access token or authentication token. You may also call your Sonar scanner tool as sonar-scanner, or anything else you choose, and don't forget to include it in your pipeline.

```
SONARCLOUD = 'Sonarcloud'
```

There are two options to run sonarcloud :

- 1) Create sonar-project-properties file in your git repository and give the sonarcloud details like give below:

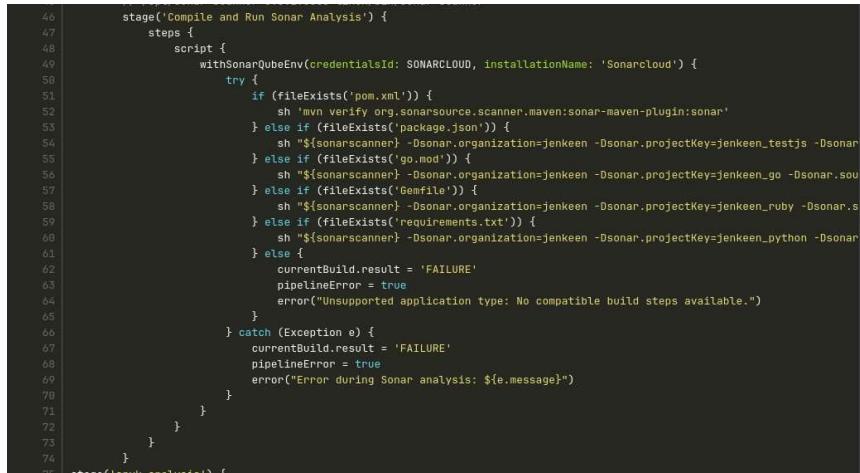


The screenshot shows a GitHub repository page for 'AnaCart / sonar-project.properties'. The file contains the following configuration:

```
1 # # SonarQube server URL
2 sonar.host.url=https://sonarcloud.io
3 sonar.projectKey=jenkeen_ancart
4 sonar.organization=jenkeen
5 sonar.projectName=ancart
6 sonar.login=1cfda1a445cdc4f50276b11dab723c83738f0398
7 sonar.sources=.
8 sonar.exclusions=node_modules/**
9 sonar.language=js
10
```

Here, adds your sonar-scanner path to the Jenkins pipeline script along with your pom.xml, csproj, solution file, package.Json, Gem file, requirement.txt, etc.

- 2) You can directly mention your sonar cloud script in the Jenkins file.



The Jenkins pipeline script includes a stage for Sonar analysis:

```
46     stage('Compile and Run Sonar Analysis') {
47         steps {
48             script {
49                 withSonarQubeEnv(credentialsId: SONARCLOUD, installationName: 'Sonarcloud') {
50                     try {
51                         if (fileExists('pom.xml')) {
52                             sh "mvn verify org.sonarsource.scanner.maven:sonar-maven-plugin:sonar"
53                         } else if (fileExists('package.json')) {
54                             sh "${sonarscanner} -Dsonar.organization=jenkeen -Dsonar.projectKey=jenkeen_testjs -Dsonar
55                         } else if (fileExists('go.mod')) {
56                             sh "${sonarscanner} -Dsonar.organization=jenkeen -Dsonar.projectKey=jenkeen_go -Dsonar.su
57                         } else if (fileExists('Gemfile')) {
58                             sh "${sonarscanner} -Dsonar.organization=jenkeen -Dsonar.projectKey=jenkeen_ruby -Dsonar.s
59                         } else if (fileExists('requirements.txt')) {
60                             sh "${sonarscanner} -Dsonar.organization=jenkeen -Dsonar.projectKey=jenkeen_python -Dsonar
61                         } else {
62                             currentBuild.result = 'FAILURE'
63                             pipelineError = true
64                             error("Unsupported application type: No compatible build steps available.")
65                         }
66                     } catch (Exception e) {
67                         currentBuild.result = 'FAILURE'
68                         pipelineError = true
69                         error("Error during Sonar analysis: ${e.message}")
70                     }
71                 }
72             }
73         }
74     }
```

- 3) If Sonar throws compile errors, ensure to edit your versioning according for a completed compile of the app. For example, maven plugin errors may require an edit of the pom.xml file that is used in the repo.

#### Stage 4 (Snyk):

Snyk is used to perform security vulnerability scans, so integrate it with Jenkins by giving it a personal access token or authentication token. You may also call your snyk Installation tool as Snyk@latest, or anything else you choose, and don't forget to include it in your pipeline.

```
SNYK_INSTALLATION = 'snyk@latest'  
SNYK_TOKEN = 'Snyk'
```

In your pipeline, now mention your installation and the name of your Snyk token so that it knows which API you are attempting to access.

```
stage('snyk_analysis') {  
    steps {  
        script {  
            echo 'Testing...'  
            try {  
                snykSecurity(  
                    snykInstallation: SNYK_INSTALLATION,  
                    snykTokenId: SNYK_TOKEN,  
                    failOnIssues: false,  
                    monitorProjectOnBuild: true,  
                    additionalArguments: '--all-projects --d'  
                )  
            } catch (Exception e) {  
                currentBuild.result = 'FAILURE'  
                pipelineError = true  
                error("Error during snyk_analysis: ${e.message}")  
            }  
        }  
    }  
}
```

## Stage 5 (Java Detection):

Java may be automatically detected, and you will be able to see whether it is supported or not. [Ensure to setup the correct JDK in Jenkins tools before doing it.](#)

```
// Define the detectJavaVersion function outside of the pipeline block
def detectJavaVersion() {
    def javaVersionOutput = sh(script: 'java -version 2>&1', returnStatus: false, returnStdout: true).trim()
    def javaVersionMatch = javaVersionOutput =~ /openjdk version "(\d+\.\d+)/

    if (javaVersionMatch) {
        def javaVersion = javaVersionMatch[0][1]

        if (javaVersion.startsWith("1.8")) {
            return '8'
        } else if (javaVersion.startsWith("11")) {
            return '11'
        } else if (javaVersion.startsWith("17")) {
            return '17'
        } else {
            error("Unsupported Java version detected: ${javaVersion}")
        }
    } else {
        error("Java version information not found in output.")
    }
}
```

So here the java detects and set java pipeline script looks like:

```
stage('Detect and Set Java') {
    steps {
        script {
            try {
                def javaVersion = detectJavaVersion()
                tool name: "Java_${javaVersion}", type: 'jdk'
                sh 'java --version'
            } catch (Exception e) {
                currentBuild.result = 'FAILURE'
                pipelineError = true
                error("Error during Java version detection: ${e.message}")
            }
        }
    }
}
```

## Stage 6 (Multi-Language Build and Deployment):

During this phase, there is a variety of programming languages, including front-end, back-end, iOS, Android, Ruby, Flask, and so forth. Based on the languages I provided, the system will identify the source code from your repository and install, build, and execute the test in accordance with the pipeline script that we previously discussed.

```
stage('Frontend Build and Test') {
    steps {
        script {
            try {
                if (fileExists('package.json')) {
                    sh 'npm install --force'
                    //sh 'npm test'
                } else {
                    echo 'No package.json found, skipping Frontend build and test.'
                }
            } catch (Exception e) {
                currentBuild.result = 'FAILURE'
                pipelineError = true
                error("Error during Frontend build and test: ${e.message}")
            }
        }
    }
}
stage('Java Spring Boot Build and Test') { [ ] }
stage('.NET Build and Test') { [ ] }
stage('PHP Build and Test') { [ ] }
stage('iOS Build and Test') { [ ] }
stage('Android Build and Test') { [ ] }
stage('Ruby on Rails Build and Test') { [ ] }
stage('Flask Build and Test') { [ ] }
stage('Django Build and Test') { [ ] }
stage('Rust Build and Test') { [ ] }
stage('Ruby Sinatra Build and Test') { //To build and run a Ruby Application }
```

The pipeline script for the multi-lang build in the image above.

## Stage 7 (Docker Build and Push):

At this Stage we are going to dockerize our project after building our source code. Our pipeline script will automatically identify whether a dockerfile exists and generate the dockerfile if not, else it will display the dockerfile not found.

**Note:** Make sure you specify the name of your Docker image correctly in the environment stage (the variable name will automatically identify and take the image name). The Docker file name is case-sensitive, add docker tool and docker API in your Jenkins.

```
stage('Build and Push Docker Image') {
    steps {
        script {
            try {
                if (fileExists('Dockerfile')) {
                    withDockerRegistry(credentialsId: DOCKER_REGISTRY_CREDENTIALS, toolName: DOCKER_TOOL, url: DOCKER_URL) {
                        def dockerImage = docker.build(DOCKER_IMAGE, ".")
                        // Push the built Docker image
                        dockerImage.push()
                    }
                } else {
                    echo "Dockerfile not found. Skipping Docker image build and push."
                }
            } catch (Exception e) {
                currentBuild.result = 'FAILURE'
                pipelineError = true
                echo "Error during Docker image build and push: ${e.message}"
            }
        }
    }
}
```

We will push and store our image on container registries such as Docker Hub, AWS ECR, GCP GCR, Harbour, and others at this stage. I used Docker Hub in this instance by supplying my credentials and indicating the Docker API that I am pushing to my hub repository. Don't forget to set up a repository on Docker Hub before that.

To link with your container registries, make sure you provide your credentials or personal access token to Jenkins. Mention your credentials in the environment stage of the pipeline.

```
environment {
    SONARCLOUD = 'Sonarcloud'
    SNYK_INSTALLATION = 'snyk@latest'
    SNYK_TOKEN = 'Snyk'
    DOCKER_REGISTRY_CREDENTIALS = 'Docker_Server'
    DOCKER_IMAGE = '████████:latest'
    DOCKER_TOOL = 'Docker'
    DOCKER_URL = 'https://index.docker.io/v1/'
    KUBE_CONFIG = 'kubernetes'
```

**Note:** By using the docker run command locally, you can verify whether your Docker image is up and running or not.

## **Stage 8 (Aqua Trivy Scan):**

Now that the Docker build has finished and our image has been successfully produced, it's time to detect any vulnerabilities by scanning it. We're going to use Aqua Trivy Scan for image Scanning.

Verify that Aqua Trivy has been installed on your local system. If you don't already have trivy installed on your system, get it from docker and run trivy image. Once it's finished, try using docker trivy image to scan your image. Put the name of your image after the image command using the following docker trivy command:

```
docker run ghcr.io/aquasecurity/trivy:latest image DOCKER_IMAGE
```

```
stage('Trivy Scan') {
    steps {
        script {
            def trivyInstalled = sh(script: 'command -v trivy', returnStatus: true) == 0
            def imageName = DOCKER_IMAGE
            if (trivyInstalled) {
                sh "trivy image --format table ${imageName}"
            } else {
                // Run trivy using Docker
                sh "docker run --rm -v /var/run/docker.sock:/var/run/docker.sock aquasec/trivy image --format table ${imageName}"
            }
        }
    }
}
```

Mention your docker image here, it will scan and detect the vulnerabilities.

### Stage 9 (Kubernetes):

It is the main stage we are now in. Up to this point, everything went according to plan as we built, deployed, and dockerized our image and pushing it to the hub. However, we must host our program in runtime. What is the process? By applying Kubernetes is the way forward.

Make sure you have installed clusters before integrating Kubernetes with Jenkins; it doesn't matter if they are minikube, kind, or kubeadm. If you're using a loadbalancer, install kubeadm and construct your master and worker nodes. If you're using nodeport, install your minikube or kind cluster on your Jenkins slave machine.

**Note:** You can integrate your Jenkins with your Kubernetes cluster using the kube config file.

```
stage('Kubernetes Deployment') {
    steps {
        script {
            def configFile = 'deployment.yaml'
            if (fileExists(configFile)) {
                kubernetesDeploy(configs: configFile, kubeconfigId: KUBE_CONFIG)
            } else {
                error("Error: $configFile does not exist")
                currentBuild.result = 'FAILURE'
                pipelineError = true
            }
        }
    }
}
```

In the environment stage, provide your kube configuration credentials and add the name of your deployment.yaml file in place of the config file.

```
environment {
    SONARCLOUD = 'Sonarcloud'
    SNYK_INSTALLATION = 'snyk@latest'
    SNYK_TOKEN = 'Snyk'
    DOCKER_REGISTRY_CREDENTIALS = 'Docker_Server'
    DOCKER_IMAGE = 'ganesharavind124/anacart:latest'
    DOCKER_TOOL = 'Docker'
    DOCKER_URL = 'https://index.docker.io/v1/'
    KUBE_CONFIG = 'kubernetes'
```

The application will now be operational on your pods after the deployment was successfully created there. You can test by running (kubectl get svc) with the service name. If you used a loadbalancer, you would receive your external IP and be able to access your application through it.

If you ran (minikube service MY-SERVICE-NAME) with minikube, you will receive your IP and port number and be able to access your application through it.

## Stage 10 (Zaproxy Testing):

We have already performed SAST Scan and Application Testing; going forward, we will perform DAST, which is intended to assist in detecting security vulnerabilities in web applications throughout the software development and testing stages.

ZAP testing will involve using that URL to test an application hosted in PROD or DEV. We'll use a variety of scan methods, including spidering, active, passive, fuzzer, proxy interception, and scripting attacks. However, for now, I'm just doing a basic zap test that generates and provides us with a report.

Make sure that ZAPROXY has installed on your Local or Instance or server systems.

With the use of a local minikube, provided the URL can be provided directly in Jenkins based on deployment.yaml configuration in the pipeline.

```
stage('Run DAST Using ZAP') {
    steps {
        script {
            try {
                def targetURL = "http://192.168.49.2:32767/" // Replace with your actual target URL
                def zapCommand = "zaproxy -cmd -quickurl ${targetURL}"
                sh(zapCommand)
                archiveArtifacts artifacts: 'zap_report.html'
            } catch (Exception e) {
                currentBuild.result = 'FAILURE'
                error("Error during ZAP DAST: ${e.message}")
            }
        }
    }
}
```

When using Loadbalancer (Cloud), the zap command will be executed automatically without the need for manual entry, and the IP and port will be generated automatically. Use the below script to detect URL automatically.

```
stage('Run DAST Using ZAP') {
    steps {
        script {
            try {
                def targetURL = sh(script: 'kubectl get service/' + serviceName + ' -o json | jq -r ".status.loadBalancer.ingress[0].hostname"', returnStatus: true).trim()
                def zapCommand = "zap.sh -cmd -quickurl http://${targetURL} -quickprogress -quickout ${ZAP_WORKSPACE}/zap_report.html"
                sh(zapCommand)
                archiveArtifacts artifacts: 'zap_report.html'
            } catch (Exception e) {
                currentBuild.result = 'FAILURE'
                error("Error during ZAP DAST: ${e.message}")
            }
        }
    }
}
```

## Sample Results:

### SNYK:

The screenshot shows the Jenkins interface with a 'Snyk Security Report' card. The report details a critical vulnerability found in the 'easybuggy' project. The vulnerability is described as 'Deserialization of Untrusted Data' and is associated with the package manager 'maven' and the module 'log4j'. The report also indicates 18 known vulnerabilities, 18 vulnerable dependency paths, and 41 dependencies.

Key details from the report:

- Scanned the following path: /home/kali/jenkins/workspace/Devescops/pom.xml (maven)
- 18 known vulnerabilities
- 18 vulnerable dependency paths
- 41 dependencies

Project: org.1245osslab.easybuggy/easybuggy Path: /home/kali/jenkins/workspace/Devescops

Package Manager: maven Manifest: pom.xml

CRITICAL SEVERITY

Deserialization of Untrusted Data

- Package Manager: maven
- Vulnerable module: log4j[log4]

Jenkins 2.426.1

### SONARCLOUD:

The screenshot shows the SonarCloud interface for the 'easybuggy' project. The main dashboard displays the 'Main Branch Status' as 'Passed' with a green checkmark icon. It also shows the 'Main Branch Evolution' over the last 32 minutes, indicating 285 findings across various categories: Bugs, Code Smells, Vulnerabilities, and Security Hotspots. The interface includes navigation links for 'My Projects', 'My Issues', 'Explore', and a search bar.

Key metrics from the SonarCloud dashboard:

- Main Branch Status: Passed
- Findings: 285
- Bugs: 57
- Code Smells: 138
- Vulnerabilities: 52
- Security Hotspots: 38

## DOCKER HUB:

The screenshot shows the Docker Hub interface for the repository `martian1337/easybuggy`. At the top, there's a navigation bar with links for Explore, Repositories (which is selected), Organizations, a search bar, and various user icons. Below the navigation, the repository path is shown as `martian1337 / Repositories / easybuggy / General`, with a note indicating 0 of 1 private repositories.

The main content area has tabs for General, Tags, Builds, Collaborators, Webhooks, and Settings. A callout box suggests adding a short description for the repository, with a link to Update.

**General Information:**

- Description:** This repository does not have a description.
- Last pushed:** 16 minutes ago.

**Docker commands:** To push a new tag to this repository, use the command `docker push martian1337/easybuggy:tagname`.

**Tags:** This repository contains 1 tag(s). The table shows one tag: `latest` (Image type, Pulled 3 days ago, Pushed 16 minutes ago).

**Automated Builds:** Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating. Available with Pro, Team and Business subscriptions. [Read more about automated builds](#).

## TRIVY:

martian1337/easybuggy:latest (debian 11.6)						
Total: 127 (UNKNOWN: 0, LOW: 73, MEDIUM: 31, HIGH: 21, CRITICAL: 2)						
Library	Vulnerability	Severity	Status	Installed Version	Fixed Version	Title
apt	CVE-2011-3374	LOW	affected	2.2.4		It was found that apt-key in apt, all versions, do not   correctly...   <a href="https://avd.aquasec.com/nvd/cve-2011-3374">https://avd.aquasec.com/nvd/cve-2011-3374</a>
bash	CVE-2022-3715	HIGH		5.1.2+deb11u1		a heap-buffer-overflow in valid_parameter_transform   <a href="https://avd.aquasec.com/nvd/cve-2022-3715">https://avd.aquasec.com/nvd/cve-2022-3715</a>
	TEMP-0841856-B18BAF	LOW				[Privilege escalation possible to other user than root]   <a href="https://security-tracker.debian.org/tracker/TEMP-0841856-B1-8BAF">https://security-tracker.debian.org/tracker/TEMP-0841856-B1-8BAF</a>
bsdutils	CVE-2022-0563			1:2.36.1-8+deb11u1		util-linux: partial disclosure of arbitrary files in chfn   and chsh when compiled...   <a href="https://avd.aquasec.com/nvd/cve-2022-0563">https://avd.aquasec.com/nvd/cve-2022-0563</a>
coreutils	CVE-2016-2781		will_not_fix	8.32.4+b1		coreutils: Non-privileged session can escape to the parent   session in chroot   <a href="https://avd.aquasec.com/nvd/cve-2016-2781">https://avd.aquasec.com/nvd/cve-2016-2781</a>
	CVE-2017-18018		affected			coreutils: race condition vulnerability in chown and chgrp   <a href="https://avd.aquasec.com/nvd/cve-2017-18018">https://avd.aquasec.com/nvd/cve-2017-18018</a>
e2fsprogs	CVE-2022-1304	HIGH		1.46.2-2		e2fsprogs: out-of-bounds read/write via crafted filesystem   <a href="https://avd.aquasec.com/nvd/cve-2022-1304">https://avd.aquasec.com/nvd/cve-2022-1304</a>
gcc-10-base	CVE-2023-4039	MEDIUM		10.2.1-6		gcc: -fstack-protector fails to guard dynamic stack   allocations on ARM64   <a href="https://avd.aquasec.com/nvd/cve-2023-4039">https://avd.aquasec.com/nvd/cve-2023-4039</a>
ncrc-9-base				9.3.0-22		

## Troubleshooting Trivy + Docker deployment:

You may encounter a permissions issue with accessing the Docker daemon like this:

```
docker: permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Post "http://%2Fvar%2Frun%2Fdocker.sock/v1.24/containers/create": dial unix /var/run/docker.sock: connect: permission denied.
```

This usually happens when the user running the Docker command does not have the necessary permissions to access the Docker daemon. Here are a few ways to resolve this issue:

**Run as Root User:** The simplest way is to run the Docker command as the root user using sudo. This gives you the necessary permissions to interact with the Docker daemon. Try running:

```
sudo docker run --rm -v /var/run/docker.sock:/var/run/docker.sock aquasec/trivy image --format table docker:latest
```

Be cautious when using sudo as it grants high-level access to your system.

**Add Your User to the Docker Group:** A more secure and commonly used method is to add your user to the docker group. This allows any user in the docker group to run Docker commands without needing root access.

First, add your user to the Docker group with: `sudo usermod -aG docker $USER`

Then, log out and log back in for this change to take effect. Alternatively, you can use newgrp docker to activate the changes in the current session.

After this, try running your Docker command again without sudo.

**Check Docker Daemon Status:** Ensure that the Docker daemon is running. You can check this with:

```
sudo systemctl status docker
```

If it's not running, start it with: `sudo systemctl start docker`

**Verify Socket Permissions:** Check the permissions of the Docker socket file: `ls -l /var/run/docker.sock`

The output should show that the file is owned by root and the docker group. If it's not, you can change the group with:

```
sudo chgrp docker /var/run/docker.sock
```

1. **Verify Group Membership:** Ensure that the user running your pipeline is indeed a member of the Docker group. You can check this with the groups command:

```
groups [username]
```

Replace [username] with the username of the pipeline's user. Ensure docker is listed in the output.

**Restart the Docker Service:** After modifying group memberships, it can sometimes be necessary to restart the Docker service for the changes to take effect. Use:

```
sudo systemctl restart docker
```

By following these steps, you should be able to resolve the permission issues and successfully run your Docker command.

## Permissions to run Docker

1. Run the below command to add user and Jenkins to the appropriate group for permissions
  - a. '`sudo usermod -aG docker $USER && sudo usermod -aG docker jenkins && newgrp docker`'
2. Restart docker service with command below
  - a. '`sudo systemctl restart docker`'
3. Set permissions on Docker socket
  - a. '`sudo chmod 666 /var/run/docker.sock`'

## Lost Jenkins Password

1. If you don't change the initial password and lose it for any reason, you can always retrieve it with the command: `sudo cat /var/lib/jenkins/secrets/initialAdminPassword`

## Too many Kubernetes pods/reset

1. If you have too many pods and want to remove them, you can run the below command to delete them all within a namespace or all namespaces with the commands here:  
`kubectl delete pods --all --namespace <name>` or `kubectl delete pods --all --all-namespaces`