



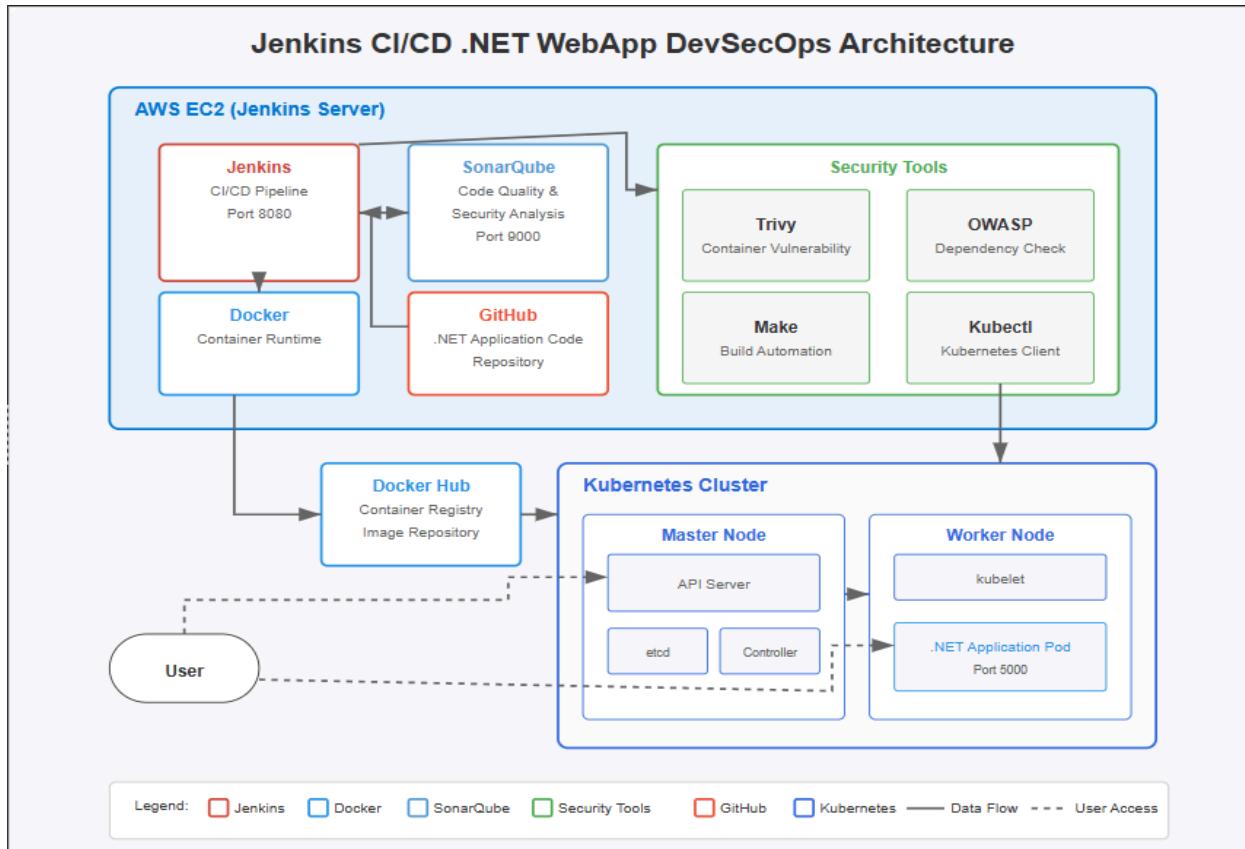
Vadakathi Muhammed Suhaib

Technical Apprentice

Emp ID: X48GRSTML

muhammed.suhaib@cprime.com

Jenkins CI/CD | .NET WebApp DevSecOps Capstone Project



Jenkins CI/CD | .NET WebApp DevSecOps Capstone Project

Introduction

This documentation provides a comprehensive guide for implementing a CI/CD pipeline for a .NET application using Jenkins. The project demonstrates a real-world DevSecOps implementation by deploying a .NET-based application using Jenkins as a CICD tool and deploying the application on Docker Containers and a Kubernetes cluster.

Tools Used

1. AWS EC2

Description: Amazon Elastic Compute Cloud (EC2) provides scalable computing capacity in the Amazon Web Services cloud.

Usage: Hosting the Jenkins server and Kubernetes nodes.

2. Jenkins

Description: An open-source automation server that enables developers to build, test, and deploy their software.

Usage: Creating and managing our CI/CD pipeline.

3. Docker

Description: A platform for developing, shipping, and running applications inside containers.

Usage: Containerizing our .NET application and running SonarQube.

4. SonarQube

Description: An open-source platform for continuous inspection of code quality.

Usage: Performing code quality and security analysis.

5. Trivy

Description: A simple and comprehensive vulnerability scanner for containers and other artifacts.

Usage: Scanning Docker images for vulnerabilities.

6. OWASP Dependency Check

Description: A utility that identifies project dependencies and checks if there are any known, publicly disclosed vulnerabilities.

Usage: Checking for vulnerabilities in project dependencies.

7. Kubernetes

Description: An open-source system for automating deployment, scaling, and management of containerized applications.

Usage: Orchestrating our containerized application.

8. Make

Description: A build automation tool that automatically builds executable programs and libraries from source code.

Usage: Simplifying Docker image building and pushing commands.

Importance of DevSecOps

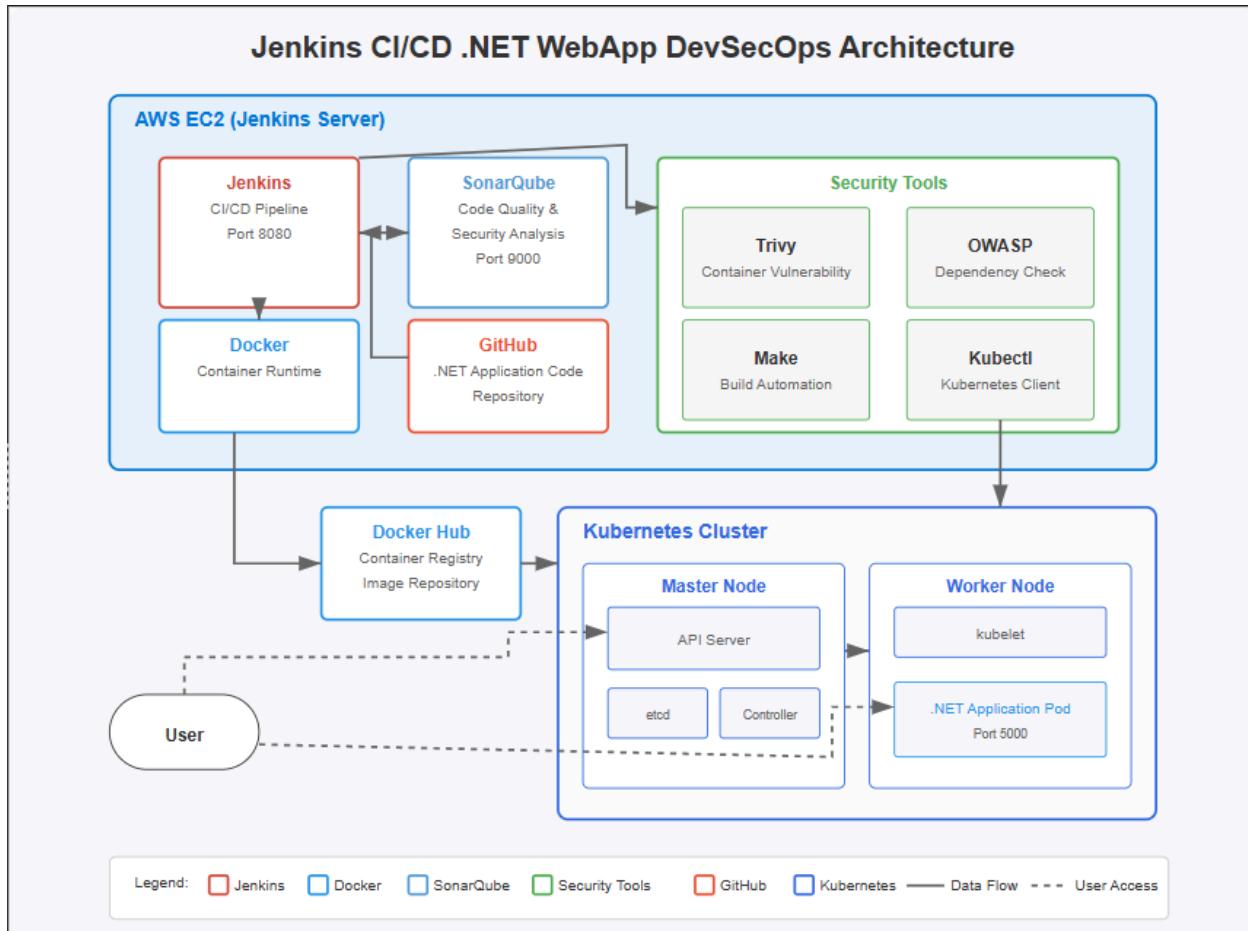
DevSecOps integrates security practices within the DevOps process. This integration is crucial because:

1. **Early Security Integration:** Security checks are performed throughout the development process, not just at the end.
2. **Continuous Security:** Regular security testing helps identify vulnerabilities early.
3. **Automated Security:** Security tests are automated, ensuring consistency and reducing human error.
4. **Compliance Management:** Helps ensure that applications meet regulatory requirements.
5. **Risk Reduction:** Reduces the risk of security breaches by identifying and addressing vulnerabilities early.

In this project, DevSecOps principles are applied through:

- SonarQube for code quality and security analysis
- Trivy for container vulnerability scanning
- OWASP Dependency Check for identifying vulnerabilities in dependencies

System Architecture



Step-by-Step Implementation Guide

Step 1: Launch an AWS T2 Large Instance

1. Use Ubuntu as the base image

The screenshot shows the AWS EC2 'Launch an instance' wizard. In the top navigation bar, 'EC2 > Instances > Launch an instance' is selected. On the left, there's a grid of AMI icons: Amazon Linux, macOS, Ubuntu, Windows, Red Hat, and SUSE Linux. The 'Ubuntu' icon is highlighted. Below the grid, the 'Amazon Machine Image (AMI)' section shows 'Ubuntu Server 24.04 LTS (HVM, SSD Volume Type)' selected. To the right, a summary box indicates 'Number of instances: 1'. The 'Software Image (AMI)' section lists 'Canonical, Ubuntu, 24.04, amd6...' and the 'Virtual server type (instance type)' is set to 't2.large'. On the left, the 'Instance type' section details the 't2.large' configuration, including its family (t2), vCPUs (2), memory (8 GiB), and current generation status. It also shows pricing information for On-Demand and On-Demand RENEWAL. A note at the bottom states 'Additional costs apply for AMIs with pre-installed software'.

2. Create a new key pair or use an existing one

3. Enable HTTP and HTTPS settings in the Security Group

The screenshot continues the 'Launch an instance' wizard. In the 'Auto-assign public IP' section, 'Enable' is selected. Below it, a note says 'Additional charges apply when outside of free tier allowance'. The 'Firewall (security groups)' section shows a 'Create security group' button is selected. The summary box on the right shows 'Number of instances: 1', 'Software Image (AMI): Canonical, Ubuntu, 24.04, amd6...', and 'Virtual server type (instance type): t2.large'. The 'Firewall (security group)' section indicates a 'New security group' is being created.

The screenshot shows the AWS EC2 'Instances' page. The left sidebar has 'EC2' selected under 'Instances'. The main area displays a table of instances. One row is highlighted for 'Jenkins-Dotnet' with instance ID 'i-04a14bfd06a286039'. The 'Instance state' is listed as 'Running'. Other columns include 'Name' (Jenkins-Dotnet), 'Instance ID' (i-04a14bfd06a286039), 'Instance type' (t2.large), 'Status check' (2/2 checks passed), and 'Alarm status' (View alarms +). The top navigation bar shows 'EC2 > Instances > Instances' and the search bar contains 'iam'.

4. Open ports 8080 (Jenkins), 9000 (SonarQube), and 5000 (Application) in the Security Group

The screenshot shows a list of security group rules. There are three new rules added:

- Rule 1: Custom TCP on port 8080, allowing 0.0.0.0/0 from Jenkins.
- Rule 2: Custom TCP on port 9000, allowing 0.0.0.0/0 from SonarQube.
- Rule 3: Custom TCP on port 5000, allowing 0.0.0.0/0 from Application.

An 'Add rule' button is visible at the bottom left.

5. Associate elastic IP

Resource type
Choose the type of resource with which to associate the Elastic IP address.

Instance
 Network interface

⚠️ If you associate an Elastic IP address with an instance that already has an Elastic IP address associated, the previously associated Elastic IP address will be disassociated, but the address will still be allocated to your account. [Learn more](#)

If no private IP address is specified, the Elastic IP address will be associated with the primary private IP address.

Instance
i-04a14bfd06a286039

Private IP address
The private IP address with which to associate the Elastic IP address.
172.31.35.118

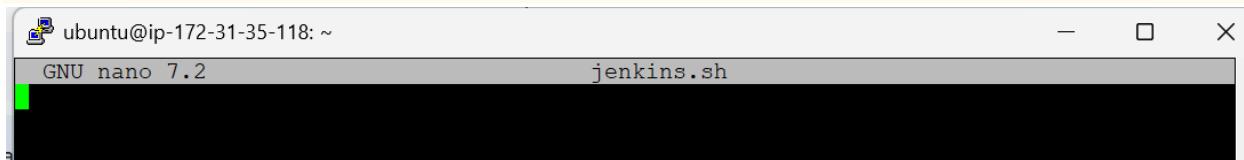
Reassociation

Step 2: Install Jenkins, Docker, and Trivy

2A - Install Jenkins

Create a script file to install Jenkins:

```
sudo nano jenkins.sh
```



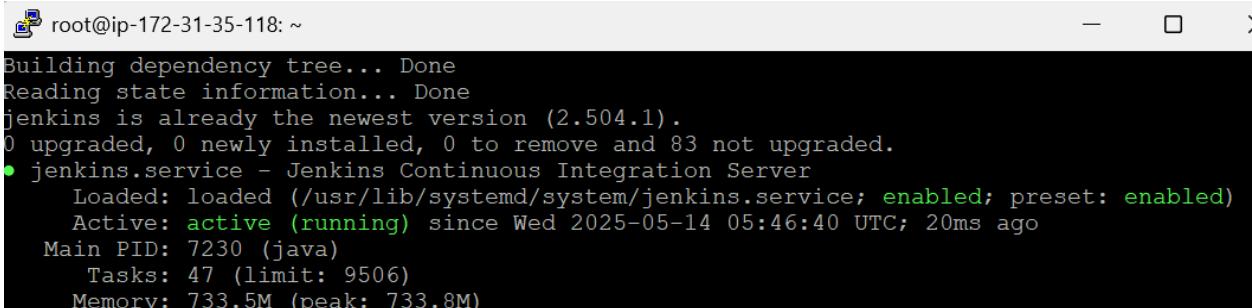
Add the following content:

```
#!/bin/bash
sudo apt update -y
wget -O -
```

```
https://packages.adoptium.net/artifactory/api/gpg/key/public |  
tee /etc/apt/keyrings/adoptium.asc  
echo "deb [signed-by=/etc/apt/keyrings/adoptium.asc]  
https://packages.adoptium.net/artifactory/deb $(awk -F=  
'/^VERSION_CODENAME/{print$2}' /etc/os-release) main" | tee  
/etc/apt/sources.list.d/adoptium.list  
sudo apt update -y  
sudo apt install temurin-17-jdk -y  
/usr/bin/java --version  
curl -fsSL  
https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo  
tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null  
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]  
https://pkg.jenkins.io/debian-stable binary/ | sudo tee  
/etc/apt/sources.list.d/jenkins.list > /dev/null  
sudo apt-get update -y  
sudo apt-get install jenkins -y  
sudo systemctl start jenkins  
sudo systemctl status jenkins
```

Make the script executable and run it:

```
sudo chmod 777 jenkins.sh  
./jenkins.sh
```

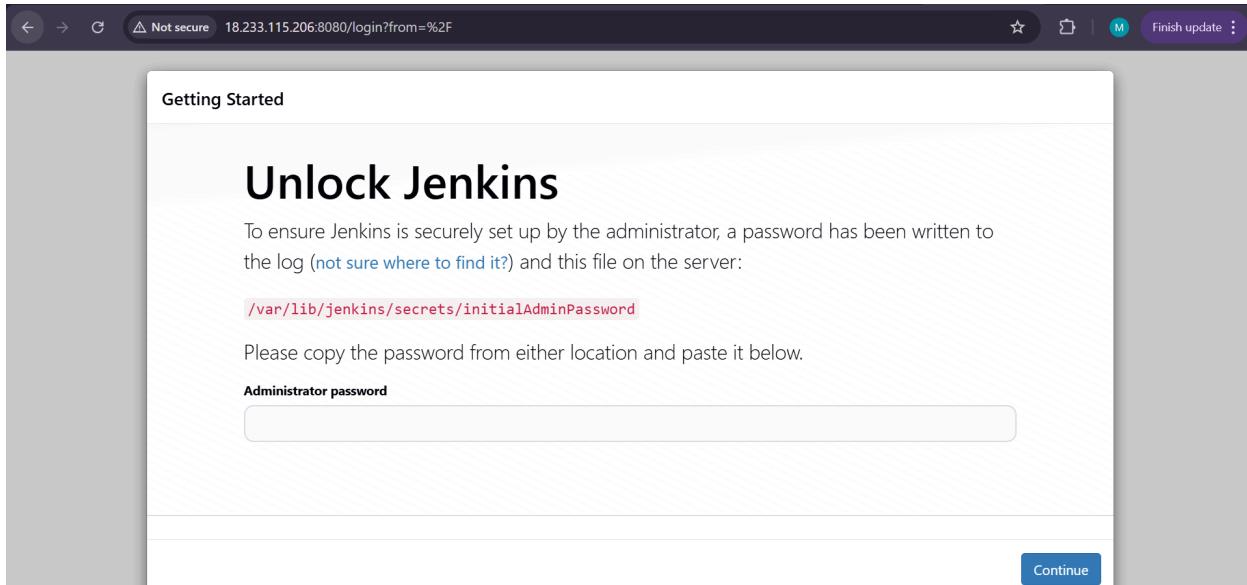


The terminal window shows the output of the Jenkins installation script. It starts with dependency building and state reading. It then indicates that Jenkins is already the newest version (2.504.1). It shows 0 upgraded, 0 newly installed, 0 to remove, and 83 not upgraded. A detailed service status for 'jenkins.service' is provided, showing it is active (running) since May 14, 2025, at 05:46:40 UTC, with a 20ms duration. The Main PID is 7230 (java), there are 47 tasks, and memory usage is 733.5M (peak: 733.8M).

```
Building dependency tree... Done  
Reading state information... Done  
jenkins is already the newest version (2.504.1).  
0 upgraded, 0 newly installed, 0 to remove and 83 not upgraded.  
● jenkins.service - Jenkins Continuous Integration Server  
    Loaded: loaded (/usr/lib/systemd/system/jenkins.service; enabled; preset: enabled)  
    Active: active (running) since Wed 2025-05-14 05:46:40 UTC; 20ms ago  
      Main PID: 7230 (java)  
        Tasks: 47 (limit: 9506)  
       Memory: 733.5M (peak: 733.8M)
```

Access Jenkins from your browser:

```
http://<EC2-Public-IP-Address>:8080  
http://18.233.115.206:8080
```



Get the Jenkins admin password:

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

```
root@ip-172-31-35-118: ~
root@ip-172-31-35-118:~# sudo cat /var/lib/jenkins/secrets/initialAdminPassword
8333fadbe0a452db62c6c7be17a3183
root@ip-172-31-35-118:~#
```

8333fadbe0a452db62c6c7be17a3183

Complete the Jenkins setup by:

1. Entering the admin password
2. Installing suggested plugins
3. Creating an admin user
4. Configuring the instance



Create First Admin User

Username: admin

Password:

Confirm password:

Instance Configuration

Jenkins URL: http://18.233.115.206:8080/

The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the BUILD_URL environment variable provided to build steps.

The proposed default value shown is **not saved yet** and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.

The screenshot shows the Jenkins dashboard at the URL <http://18.233.115.206:8080>. The top navigation bar includes a 'Not secure' warning, the Jenkins logo, and user information for 'suhail'. The dashboard menu on the left lists 'New Item', 'Build History', 'Manage Jenkins', and 'My Views'. The main content area displays the 'Welcome to Jenkins!' message: 'This page is where your Jenkins jobs will be displayed. To get started, you can set up distributed builds or start building a software project.'

2B - Install Docker

Install Docker and run a SonarQube container:

```
sudo apt-get update
sudo apt-get install docker.io -y
sudo usermod -aG docker $USER
newgrp docker
sudo chmod 777 /var/run/docker.sock
docker version
```

```
root@ip-172-31-35-118:~# sudo apt-get update
sudo apt-get install docker.io -y
sudo usermod -aG docker $USER
newgrp docker
sudo chmod 777 /var/run/docker.sock
sudo docker ps
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Hit:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease
Ign:4 https://pkg.jenkins.io/debian-stable binary/ InRelease
Hit:5 https://pkg.jenkins.io/debian-stable binary/ Release
```

```
root@ip-172-31-35-118:~# docker version
Client:
Version:          26.1.3
API version:      1.45
Go version:       go1.22.2
Git commit:       26.1.3-0ubuntu1~24.04.1
Built:            Mon Oct 14 14:29:26 2024
OS/Arch:          linux/amd64
Context:          default

Server:
Engine:
Version:          26.1.3
API version:      1.45 (minimum version 1.24)
Go version:       go1.22.2
Git commit:       26.1.3-0ubuntu1~24.04.1
```

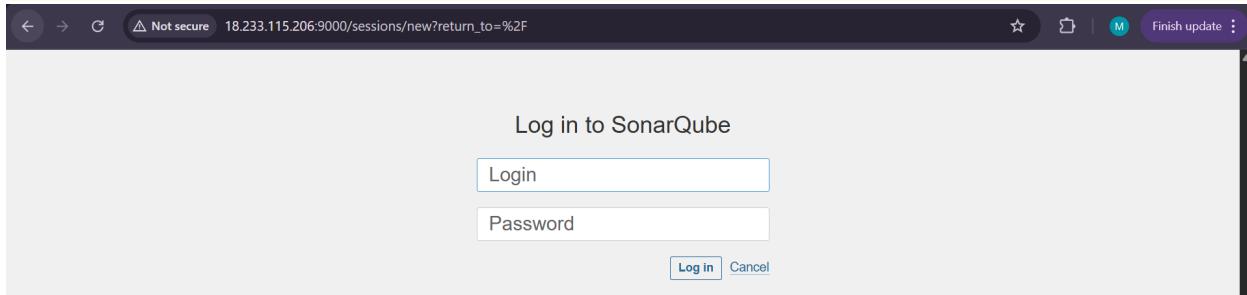
docker run -d --name sonar -p 9000:9000 sonarqube:lts-community

```
root@ip-172-31-35-118:~# docker run -d --name sonar -p 9000:9000 sonarqube:lts-community
Unable to find image 'sonarqube:lts-community' locally
lts-community: Pulling from library/sonarqube
215ed5a63843: Pull complete
094bfccb4db7a: Pull complete
4df791be4da6: Pull complete
97a8e80e60c2: Pull complete
7be47dbb02a7: Pull complete
61c8ff67f948: Pull complete
2737e808f9c3: Pull complete
4f4fb700ef54: Pull complete
Digest: sha256:93f94e0ea6148cd02d52378049dad85d0f2d6c90c485578317f76addf2af9f3d
Status: Downloaded newer image for sonarqube:lts-community
0e7e47e937ef69b1dc61726d0234ac5c130e172f224f31aee7f3f12af51b302
```

Access SonarQube from your browser:

http://<EC2-Public-IP-Address>:9000

http://18.233.115.206:9000



Default credentials:

- Username: admin
- Password: admin

The image shows the SonarQube login page. It has a light gray header with the text "Log in to SonarQube". Below the header are two input fields: the first contains the text "admin" and the second contains the text "*****". At the bottom of the form are two buttons: "Log in" and "Cancel".

After logging in, change the default password.(root)

The image shows the "Update your password" page. The title is "Update your password". A message at the top states: "This account should not use the default password." Below this, there is a section titled "Enter a new password" with the instruction "All fields marked with * are required". There are three input fields: "Old Password *", "New Password *", and "Confirm Password *". Each field has four dots in it. At the bottom right is a blue "Update" button.

The image shows the "Projects" creation screen. The URL in the browser bar is "18.233.115.206:9000/projects/create". The page has a dark header with the SonarQube logo and navigation links: Projects, Issues, Rules, Quality Profiles, Quality Gates, Administration. On the right, there is a search bar with the placeholder "Search for projects..." and a green button with the letter "A".

The main content area asks "How do you want to create your project?". It provides instructions: "Do you want to benefit from all of SonarQube's features (like repository import and Pull Request decoration)? Create your project from your favorite DevOps platform. First, you need to set up a DevOps platform configuration." Below this, there are five cards with icons and labels:

- From Azure DevOps** (Icon: Azure logo)
Set up global configuration
- From Bitbucket Server** (Icon: Bitbucket logo)
Set up global configuration
- From Bitbucket Cloud** (Icon: Bitbucket logo)
Set up global configuration
- From GitHub** (Icon: GitHub logo)
Set up global configuration
- From GitLab** (Icon: GitLab logo)
Set up global configuration

At the bottom, there is a note: "Are you just testing or have an advanced use-case? Create a project manually." followed by a "Create project" button with a double arrow icon.

2C - Install Trivy

Install Trivy vulnerability scanner:

```
sudo apt-get install wget apt-transport-https gnupg lsb-release -y
wget -qO -
https://aquasecurity.github.io/trivy-repo/deb/public.key | gpg --dearmor | sudo tee /usr/share/keyrings/trivy.gpg > /dev/null
echo "deb [signed-by=/usr/share/keyrings/trivy.gpg]
https://aquasecurity.github.io/trivy-repo/deb $(lsb_release -sc) main" | sudo tee -a /etc/apt/sources.list.d/trivy.list
sudo apt-get update
sudo apt-get install trivy -y
trivy version
```

```
root@ip-172-31-35-118:~# trivy version
Version: 0.62.1
root@ip-172-31-35-118:~# █
```

Step 3: Configure Jenkins Plugins and Tools

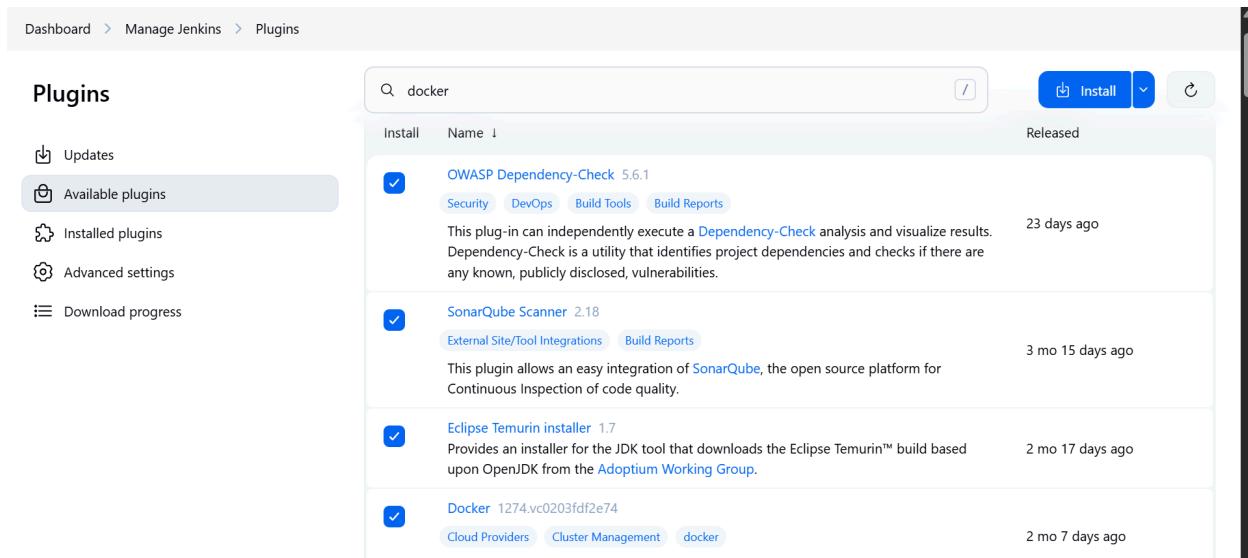
3A - Install Required Plugins

Go to Manage Jenkins → Plugins → Available Plugins and install:

The screenshot shows the Jenkins Manage Jenkins page. On the left, there's a sidebar with links like 'New Item', 'Build History', 'Manage Jenkins' (which is highlighted), and 'My Views'. Below that are sections for 'Build Queue' (no builds in the queue) and 'Build Executor Status' (0/2). The main area is titled 'Manage Jenkins' and contains several cards: 'System Configuration' (System icon, Configure global settings and paths.), 'Tools' (Wrench icon, Configure tools, their locations and automatic installers.), 'Plugins' (Gear icon, Add, remove, disable or enable plugins that can extend the functionality of Jenkins.), and 'Nodes' (Monitor icon, Add, remove, control and monitor the various nodes that Jenkins runs jobs on.). A search bar at the top right says 'Search settings'.

1. OWASP Dependency-Check
2. SonarQube Scanner
3. Eclipse Temurin Installer

4. Docker plugins (Docker, Docker Commons, Docker Pipeline, Docker API, docker-build-step)



The screenshot shows the Jenkins 'Plugins' page. The search bar at the top contains the text 'docker'. Below the search bar, there are two tabs: 'Install' and 'Name ↴'. A blue 'Install' button is visible. To the right, a 'Released' filter is applied. The results list four plugins:

- OWASP Dependency-Check 5.6.1** (Released 23 days ago): This plugin can independently execute a Dependency-Check analysis and visualize results. It identifies project dependencies and checks for known, publicly disclosed, vulnerabilities.
- SonarQube Scanner 2.18** (Released 3 mo 15 days ago): This plugin allows an easy integration of SonarQube, the open source platform for Continuous Inspection of code quality.
- Eclipse Temurin installer 1.7** (Released 2 mo 17 days ago): Provides an installer for the JDK tool that downloads the Eclipse Temurin™ build based upon OpenJDK from the Adoptium Working Group.
- Docker 1274.vc0203fdf2e74** (Released 2 mo 7 days ago): This plugin provides Cloud Providers, Cluster Management, and Docker integration.

Restart after installation



 Jenkins is restarting

3B - Configure Java in Global Tool Configuration

Go to Manage Jenkins → Tools:

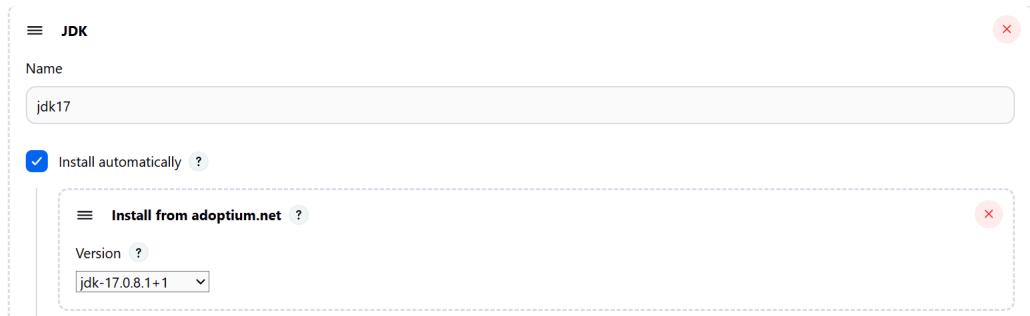
JDK installations

Add JDK

1. Add JDK

Name: jdk17

- Install automatically: Check
- Select "Install from adoptium.netr"
- Version: JDK-17



2. Add SonarQube Scanner

Name: sonar-scanner

- Install automatically: Check
- Version: Latest



3. Add OWASP Dependency-Check

Name: OWASP-Dependency-Check

- Install automatically: Check
- Version: Latest

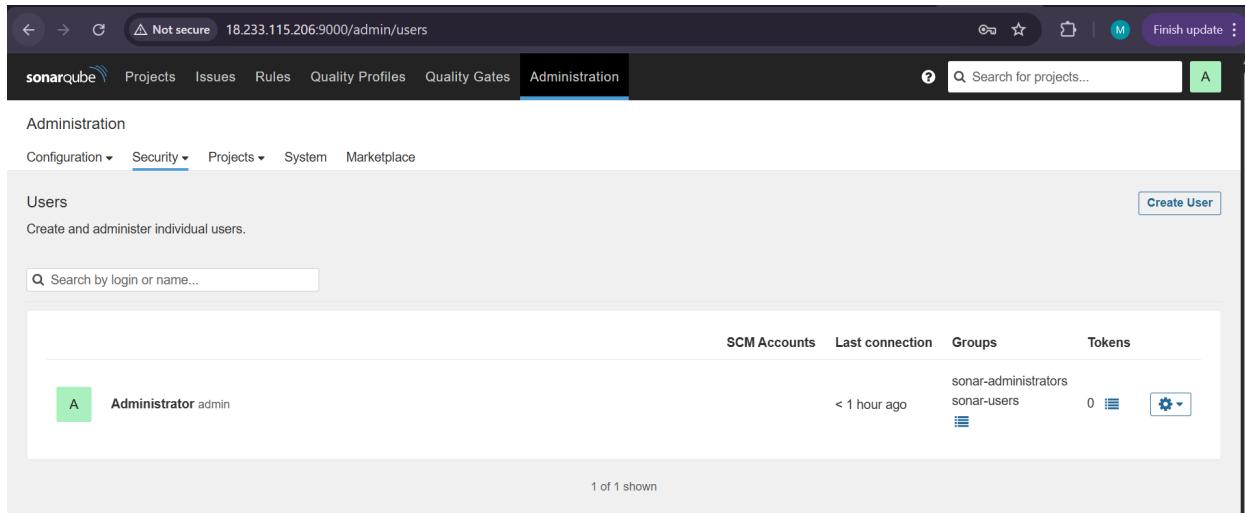


4. Apply and Save

Step 4: Configure SonarQube Server

1. Create a token in SonarQube:

- Go to SonarQube → Administration → Security → Users



Administration

Configuration ▾ Security ▾ Projects ▾ System Marketplace

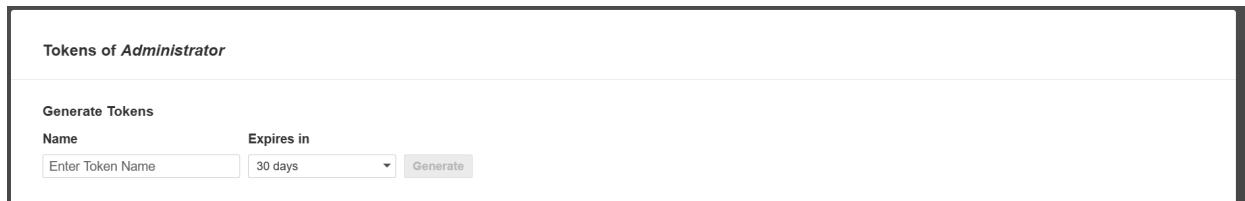
Users
Create and administer individual users.

Search by login or name...

	SCM Accounts	Last connection	Groups	Tokens
A Administrator admin		< 1 hour ago	sonar-administrators sonar-users	0 

1 of 1 shown

- Click on Tokens and Update Token



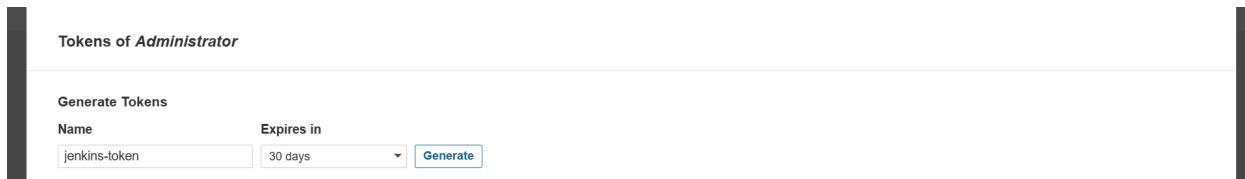
Tokens of Administrator

Generate Tokens

Name Expires in

Enter Token Name 30 days

- Give it a name and click on Generate Token



Tokens of Administrator

Generate Tokens

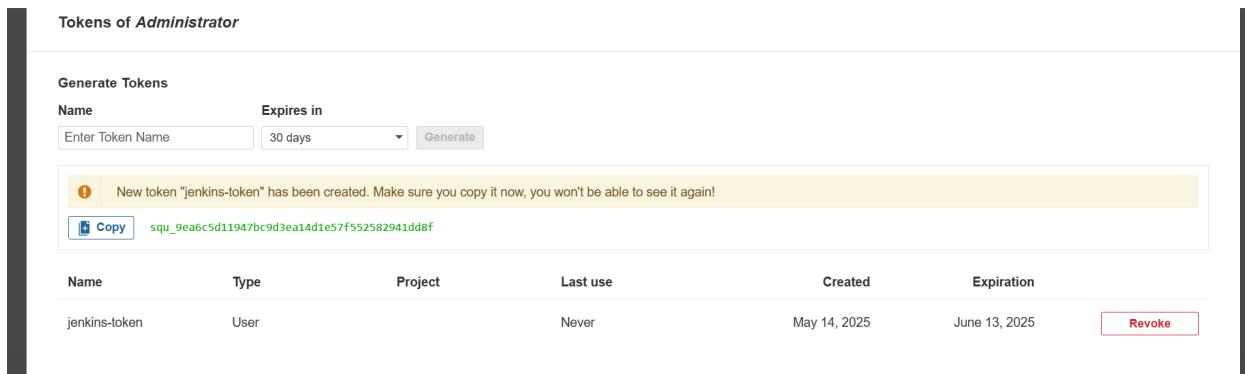
Name Expires in

jenkins-token 30 days

! New token "jenkins-token" has been created. Make sure you copy it now, you won't be able to see it again!

`squ_9ea6c5d11947bc9d3ea14d1e57f552582941dd8f`

- Copy the generated token



Tokens of Administrator

Generate Tokens

Name Expires in

Enter Token Name 30 days

! New token "jenkins-token" has been created. Make sure you copy it now, you won't be able to see it again!

`squ_9ea6c5d11947bc9d3ea14d1e57f552582941dd8f`

Name	Type	Project	Last use	Created	Expiration
jenkins-token	User		Never	May 14, 2025	June 13, 2025

squ_9ea6c5d11947bc9d3ea14d1e57f552582941dd8f

2. Add the token to Jenkins:

- Go to Jenkins → Manage Jenkins → Credentials → System → Global Credentials → Add Credentials → Secret Text

The screenshot shows the Jenkins 'Credentials' page under 'Manage Jenkins'. The 'Stores scoped to Jenkins' section is visible, showing a single entry for 'System' with '(global)' selected. The page includes a navigation bar at the top and a breadcrumb trail at the bottom.

The screenshot shows the 'Global credentials (unrestricted)' page under 'System' in 'Manage Jenkins'. It displays a table with one row for 'Global credentials (unrestricted)'. A message at the bottom encourages adding more credentials. The page includes a navigation bar at the top and a breadcrumb trail at the bottom.

- ID: sonar-token
- Description: sonar-token
- Secret: Paste the token you copied
- Click on Create

Kind

Secret text

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

Secret
.....

ID ?
sonar-token

Description ?
sonar-token

Jenkins

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

Global credentials (unrestricted)

+ Add Credentials

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

ID	Name	Kind	Description
sonar-token	sonar-token	Secret text	sonar-token

3. Configure SonarQube in Jenkins:

- Go to Jenkins → Manage Jenkins → System

Build Executor Status 0/2 ▾

System Configuration

System
Configure global settings and paths.

Tools
Configure tools, their locations and automatic installers.

- Scroll down to SonarQube servers
- Click on Add SonarQube

Dashboard > 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: sonar-server
- Server URL: http://<EC2-Public-IP>:9000

- Server authentication token: Select the sonar-token credential
- Apply and Save

The screenshot shows the SonarQube configuration interface for creating a new webhook. The 'Name' field is set to 'sonar-server'. The 'Server URL' field contains 'http://18.233.115.206:9000'. In the 'Server authentication token' dropdown, 'sonar-token' is selected. A red 'X' icon is visible in the top right corner of the form.

4. Configure Webhook in SonarQube:

- Go to SonarQube → Administration → Configuration → Webhooks

The screenshot shows the SonarQube administration interface. The 'Administration' tab is selected. Under 'Configuration', the 'Webhooks' option is highlighted. A sub-menu for 'Webhooks' is open, showing 'General Settings', 'Encryption', and 'Webhooks'. A 'Create User' button is visible in the top right corner of the main content area.

- Click on Create

The screenshot shows the 'Webhooks' configuration page. The 'Create' button is visible in the top right corner. Below it, a message states 'No webhook defined.'

- Name: Jenkins Webhook
- URL: <http://<Jenkins-Public-IP>:8080/sonarqube-webhook/>

The screenshot shows the 'Create Webhook' dialog box. The 'Name' field is filled with 'Jenkins-Webhook' and has a green checkmark. The 'URL' field is filled with 'http://18.233.115.206:8080/sonarqube-webhook/' and also has a green checkmark. A note below the URL field says 'Server endpoint that will receive the webhook payload, for example:'.

- Click on Create

5. Create a Quality Gate in SonarQube:

- Go to SonarQube → Quality Gates
- Click on Create

- Give it a name and set appropriate conditions

- Make it the default quality gate

Step 5: Install Make Package

Install the make package:

```
sudo apt install make
# to check version install or not
make -v
```

```
ubuntu@ip-172-31-35-118:~$ make -v
GNU Make 4.3
Built for x86_64-pc-linux-gnu
Copyright (C) 1988-2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
ubuntu@ip-172-31-35-118:~$
```

Step 6: Configure Docker Credentials in Jenkins

1. Install Docker plugins in Jenkins:
 - Docker
 - Docker Commons
 - Docker Pipeline
 - Docker API
 - docker-build-step

The screenshot shows the Jenkins 'Manage Jenkins' interface under the 'Plugins' section. A search bar at the top contains the text 'docker'. Below the search bar, there are tabs for 'Available plugins', 'Installed plugins', and 'Advanced settings'. The 'Available plugins' tab is selected. A list of plugins is displayed, with several checkboxes checked next to their names. The checked plugins are: 'OWASP Dependency-Check 5.6.1', 'SonarQube Scanner 2.18', 'Eclipse Temurin installer 1.7', and 'Docker 1274.vc0203fdf2e74'. Each plugin entry includes a brief description, its release date, and a link to its details page.

Name	Released
OWASP Dependency-Check 5.6.1	23 days ago
SonarQube Scanner 2.18	3 mo 15 days ago
Eclipse Temurin installer 1.7	2 mo 17 days ago
Docker 1274.vc0203fdf2e74	2 mo 7 days ago

2. Add DockerHub credentials:
 - Go to Jenkins → Manage Jenkins → Credentials → Add Credentials

The screenshot shows the Jenkins 'Credentials' page. At the top, there is a navigation bar with icons for search, protection, user suhaib, and log out. Below the navigation is a breadcrumb trail: Dashboard > Manage Jenkins > Credentials. The main title is 'Credentials'. A table header row includes columns for T, P, Store (sorted by ID), Domain, ID, and Name. Below the table, a section titled 'Stores scoped to Jenkins' lists one store: 'System' (global). The table data row for this store includes columns for P, Store (labeled '1'), Domains, ID, and Name.

- Kind: Username with password
- ID: docker
- Username: Your DockerHub username
- Password: Your DockerHub password
- Click on Create

The screenshot shows the 'Create Global Credential' form. The 'Kind' dropdown is set to 'Username with password'. The 'Scope' dropdown is set to 'Global (Jenkins, nodes, items, all child items, etc)'. The 'Username' field contains 'suhaiemd'. The 'Treat username as secret' checkbox is unchecked. The 'Password' field contains a masked value '*****'. The form has fields for 'Kind', 'Scope', 'Username', 'Treat username as secret', and 'Password'.

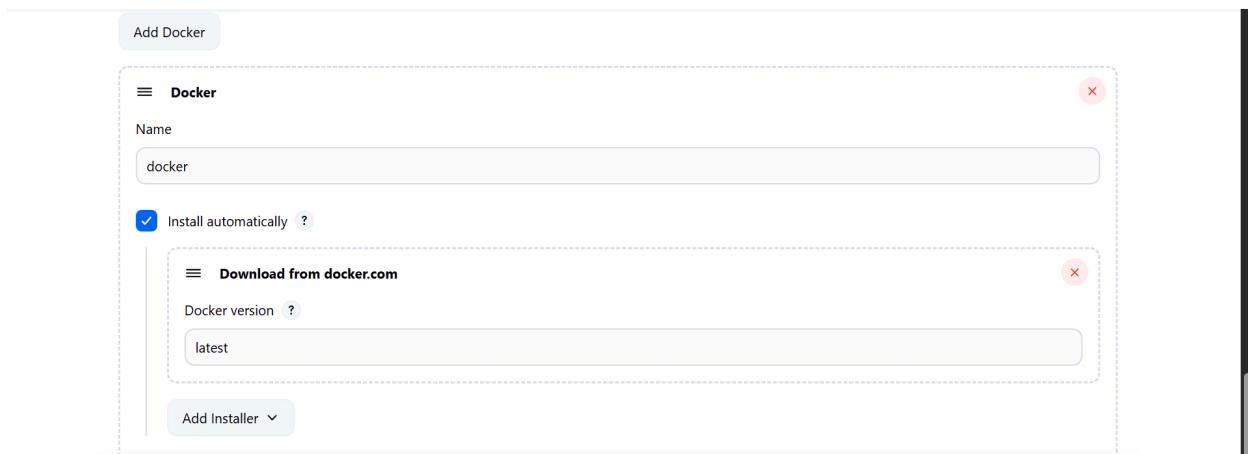
The screenshot shows the 'Global credentials (unrestricted)' page. At the top, there is a 'Add Credentials' button. Below it, a message says 'Credentials that should be available irrespective of domain specification to requirements matching.' A table lists the credentials:

ID	Name	Kind	Description
sonar-token	sonar-token	Secret text	sonar-token
docker	suhaiemd/*****	Username with password	

At the bottom, there is a note 'Icon: S M L'.

3. Install Docker Tool:

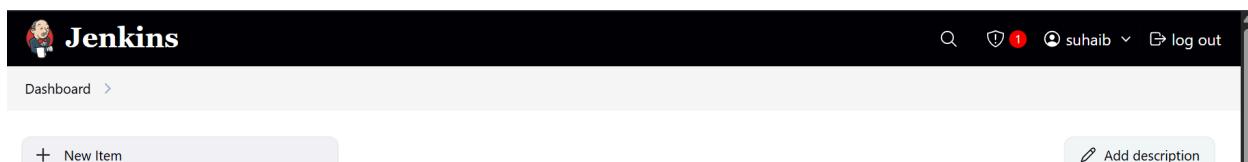
- Go to Jenkins → Manage Jenkins → Tools→ Docker and Install it



Step 7: Create a Jenkins Pipeline

Create a new Pipeline job in Jenkins:

1. Click on "New Item"



2. Enter "Dotnet-CI-CD" as the name
3. Select "Pipeline" and click OK

New Item

Enter an item name

Dotnet-CI-CD

Select an item type



Freestyle project

Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.



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.

OK

4. In the Pipeline section, select "Pipeline script" and add the following script:

```
pipeline{
    agent any
    tools{
        jdk 'jdk17'
    }
    environment {
        SCANNER_HOME=tool 'sonar-scanner'
    }
    stages {
        stage('clean workspace'){
            steps{
                cleanWs()
            }
        }
        stage('Checkout From Git'){
            steps{
                git branch: 'main', url:
'https://github.com/suhaib-md/Jenkins-CI-CD-.NET-WebApp-DevSecOps'
            }
        }
        stage("Sonarqube Analysis"){
            steps{
                withSonarQubeEnv('sonar-server') {
                    sh ''' $SCANNER_HOME/bin/sonar-scanner
-Dsonar.projectName=Dotnet-Webapp
-Dsonar.projectKey=Dotnet-Webapp '''
                }
            }
        }
        stage("quality gate"){
            steps {
                script {
                    waitForQualityGate abortPipeline: false,

```

```
credentialsId: 'sonar-token'
        }
    }
}
stage("TRIVY File scan"){
    steps{
        sh "trivy fs . > trivy-fs_report.txt"
    }
}
stage("OWASP Dependency Check"){
    steps{
        dependencyCheck additionalArguments: '--scan ./ --format XML', odcInstallation: 'OWASP-Dependency-Check'
        dependencyCheckPublisher pattern:
'**/dependency-check-report.xml'
    }
}
stage("Docker Build & tag"){
    steps{
        script{
            withDockerRegistry(credentialsId: 'docker',
toolName: 'docker'){
                sh "make image"
            }
        }
    }
}
stage("TRIVY"){
    steps{
        sh "trivy image
sevenajay/dotnet-monitoring:latest > trivy-image-report.txt"
    }
}
stage("Docker Push"){
    steps{
        script{
```

```

        withDockerRegistry(credentialsId: 'docker',
toolName: 'docker'){
            sh "make push"
        }
    }
}
stage("Deploy to container"){
    steps{
        sh "docker run -d --name dotnet -p 5000:5000
sevenajay/dotnet-monitoring:latest"
    }
}
}
}

```

Configure

Define your Pipeline using Groovy directly or pull it from source control.

Definition

Pipeline script

Script ?

```

3 v     tools{
4       jdk 'jdk17'
5   }
6 v   environment {
7       SCANNER_HOME_tool 'sonar-scanner'
8   }

```

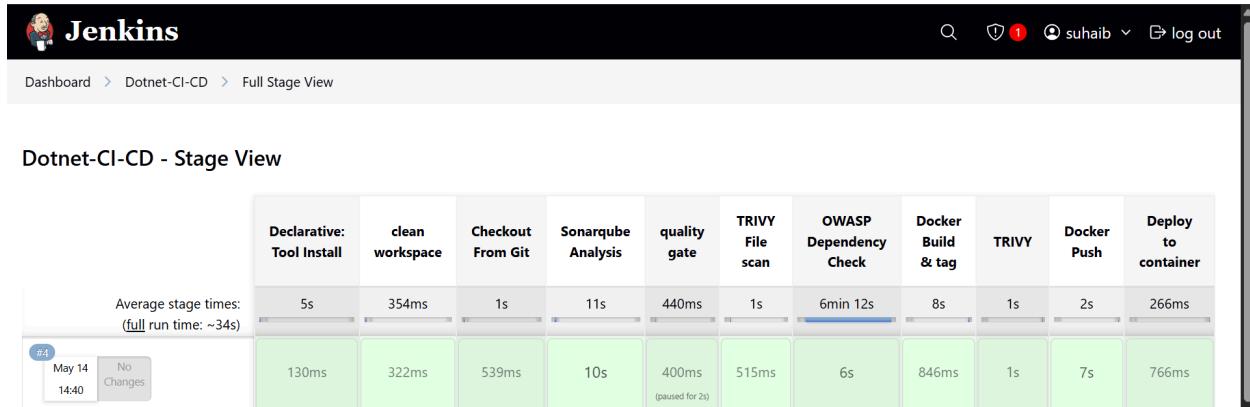
try sample Pipeline... ▾

Click on Build now and see if the pipeline works till now

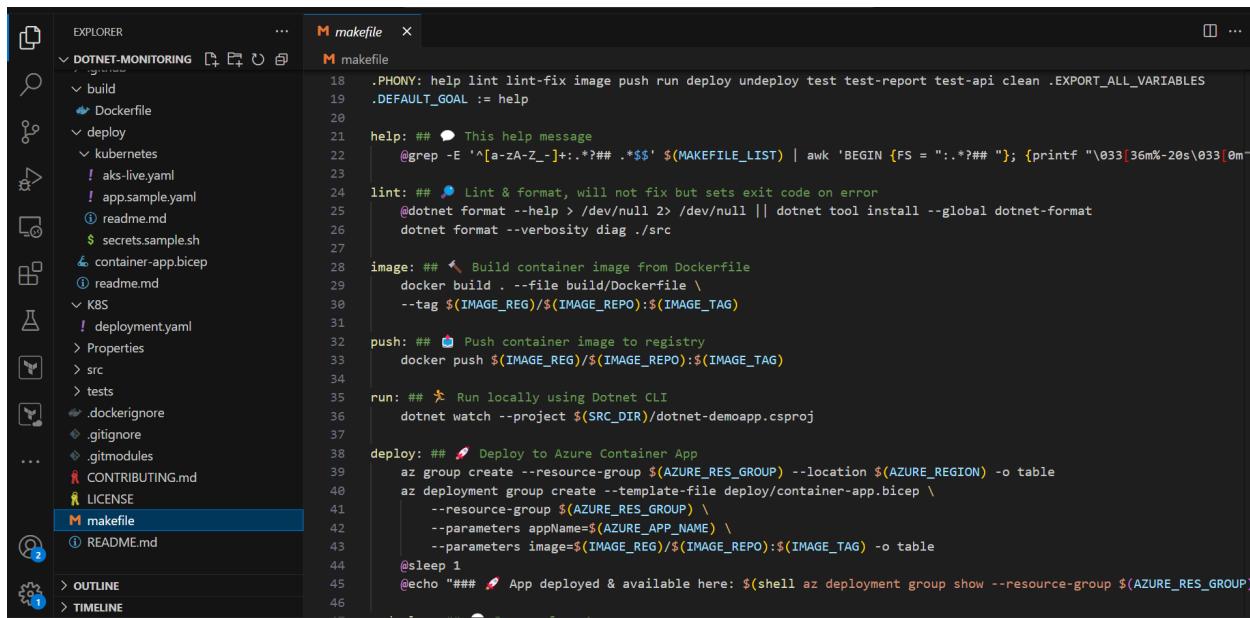
The Jenkins interface shows the status of the 'Dotnet-CI-CD' pipeline. The pipeline has failed at the 'Deploy to container' stage, indicated by a red circle with a minus sign. The stages completed successfully are: clean workspace (392ms), Checkout From Git (2s), Sonarqube Analysis (12s), quality gate (543ms), TRIVY File scan (2s), OWASP Dependency Check (1min 5s), Docker Build & tag (149ms), TRIVY (99ms), Docker Push (98ms), and Deploy to container (2min 10s). The 'Deploy to container' stage is highlighted in blue, showing a duration of 2min 10s.

clean workspace	Checkout From Git	Sonarqube Analysis	quality gate	TRIVY File scan	OWASP Dependency Check	Docker Build & tag	TRIVY	Docker Push	Deploy to container
392ms	2s	12s	543ms	2s	1min 5s	149ms	99ms	98ms	2min 10s

Wait for the pipeline to get executed



In the makefile, we already defined some conditions to build, tag and push images to dockerhub.



The screenshot shows a Visual Studio Code editor window. The left sidebar displays a file tree with various configuration files like Dockerfile, kubernetes, and bicep files. The main editor area is titled 'makefile' and contains the following content:

```
18 .PHONY: help lint lint-fix image push run deploy undeploy test test-report test-api clean .EXPORT_ALL_VARIABLES
19 .DEFAULT_GOAL := help
20
21 help: ## 📖 This help message
22     @grep -E '^[a-zA-Z_-]+:[.*?## .*$' $(MAKEFILE_LIST) | awk 'BEGIN {FS = "..*?## "}; {printf "\033[36m%20s\033[0m"}'
23
24 lint: ## 🔍 Lint & format, will not fix but sets exit code on error
25     @dotnet format --help > /dev/null 2> /dev/null || dotnet tool install --global dotnet-format
26     dotnet format --verbosity diag ./src
27
28 image: ## 🏢 Build container image from Dockerfile
29     docker build . --file build/Dockerfile \
30     --tag ${IMAGE_REG}/${IMAGE_REPO}:${IMAGE_TAG}
31
32 push: ## 🚀 Push container image to registry
33     docker push ${IMAGE_REG}/${IMAGE_REPO}:${IMAGE_TAG}
34
35 run: ## 🛠 Run locally using Dotnet CLI
36     dotnet watch --project ${SRC_DIR}/dotnet-demoapp.csproj
37
38 deploy: ## 🚤 Deploy to Azure Container App
39     az group create --resource-group ${AZURE_RES_GROUP} --location ${AZURE_REGION} -o table
40     az deployment group create --template-file deploy/container-app.bicep \
41         --resource-group ${AZURE_RES_GROUP} \
42         --parameters appName=${AZURE_APP_NAME} \
43         --parameters image=${IMAGE_REG}/${IMAGE_REPO}:${IMAGE_TAG} -o table
44     @sleep 1
45     @echo "## 🚤 App deployed & available here: $(shell az deployment group show --resource-group ${AZURE_RES_GROUP})"
46
```

That's why we are using make image and make a push in the place of docker build -t and docker push

We can see that the image is created in docker hub

A screenshot of the Docker Hub profile page for Muhammed Suhail. The profile picture is a blue circle with a white letter 'S'. The name 'Muhammed Suhail' is displayed with a small user icon. Below the profile, there are two tabs: 'Repositories' (which is selected) and 'Starred'. A search bar shows 'Search by repository na'. Below the search bar, it says 'Displaying 1 to 2 of 2 repositories'. There is one repository listed: 'suhaibmdv/dotnet-monitoring' by 'suhaibmdv' updated 9 minutes ago. The repository has 3 stars and 0 forks.

We can get the details in the SonarQube Projects page

A screenshot of the SonarQube Projects page. The top navigation bar includes 'Projects' (selected), 'Issues', 'Rules', 'Quality Profiles', 'Quality Gates', and 'Administration'. A search bar at the top right says 'Search for projects...'. On the left, there are filters for 'My Favorites' (selected) and 'All'. Below the filters, there are sections for 'Quality Gate' (Passed: 1, Failed: 0) and 'Reliability' (A rating: 0, B rating: 0, C rating: 1). The main area shows '1 project(s)' named 'Dotnet-Webapp' with a 'Passed' status. It provides a summary of analysis results: Bugs (C), Vulnerabilities (A), Hotspots Reviewed (E), Code Smells (A), Coverage (0%), Duplications (0.0%), and Lines (554, HTML, CS...).

We can access the application on port 5000 of the ip address
<public-ip of jenkins:5000>

A screenshot of a web browser displaying the '.NET Demo Web App'. The address bar shows 'Not secure 18.233.115.206:5000'. The page has a purple header with the '.NET' logo and the text '.NET Demo App', 'Info', 'Tools', and 'Monitor'. The main content area has a blue header with the text '.NET Demo Web App'. Below it, a paragraph states: 'This is a modern .NET web app using the new minimal hosting model, and Razor pages. It has been designed with cloud demos & containers in mind, and for ease of deployment into Azure or Kubernetes. A simple app to use anytime you want something lightweight to run & deploy.' Underneath, a section titled 'Basic features:' lists: 'System status / information view', 'Support for user sign-in with Azure AD and Graph API', and 'App Insights support'.

Not secure 18.233.115.206:5000/info

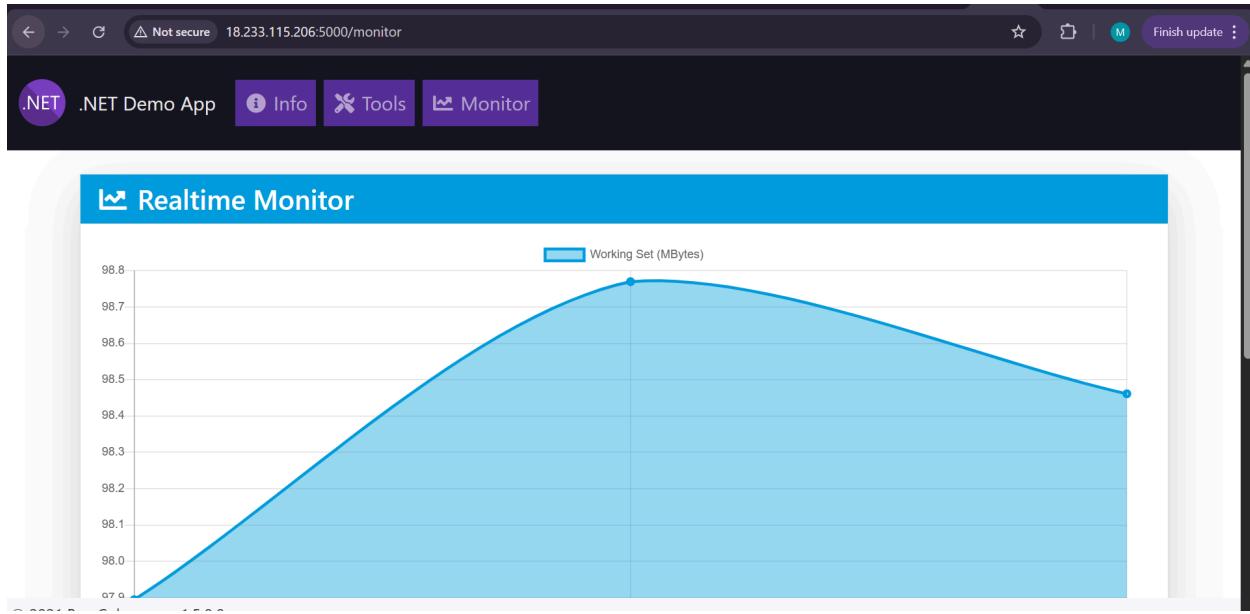
.NET Demo App Info Tools Monitor

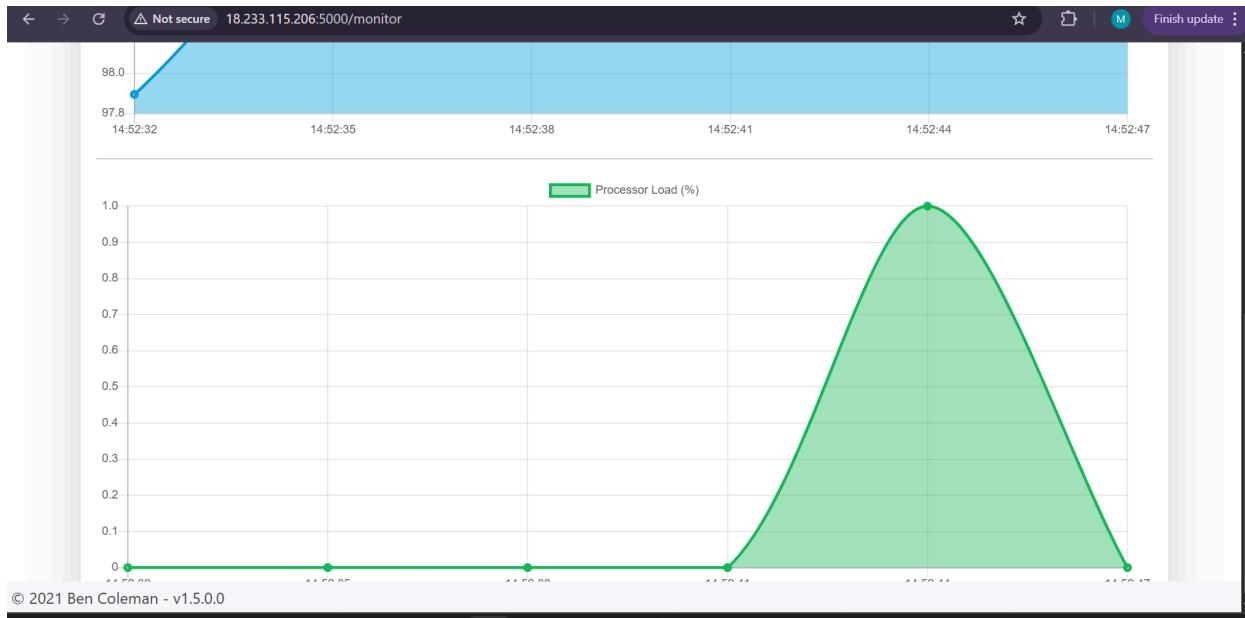
System Information

Basic Details

Containerized:	Looks like we're running in a Docker container! 😊
Kubernetes:	Not running in Kubernetes 🙁
App Insights:	Not enabled
Hostname:	2d5bcea9a750
OS Details:	Linux 6.8.0-1024-aws
Architecture:	X64 - 2 cores
Framework:	.NET 6.0.36

© 2021 Ben Coleman - v1.5.0.0





Step 8: Setup Kubernetes Cluster

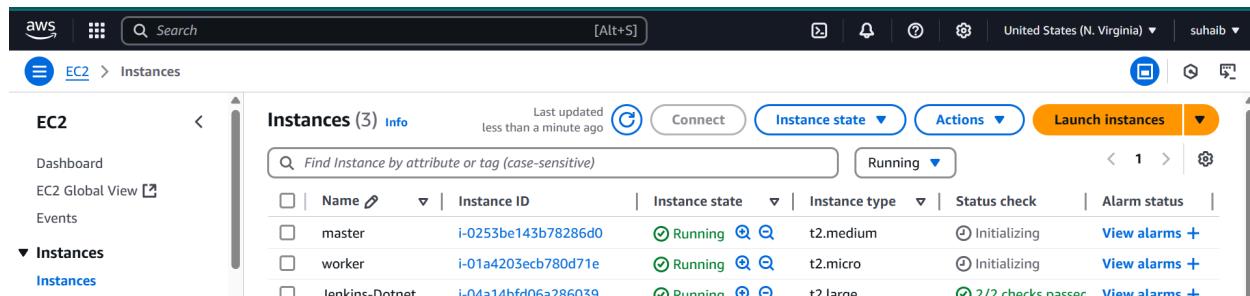
Step 8.1: Install kubectl on Jenkins Server

```
sudo apt update
sudo apt install curl
curl -LO https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl
sudo install -o root -g root -m 0755 kubectl
/usr/local/bin/kubectl
kubectl version --client
```

```
Reading state information... Done
44 packages can be upgraded. Run 'apt list --upgradable' to see them.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
curl is already the newest version (8.5.0-2ubuntu10.6).
curl set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 44 not upgraded.
   % Total    % Received % Xferd  Average Speed   Time   Time     Current
          Dload  Upload   Total Spent  Left  Speed
100  138  100  138    0      0  2107      0 --:--:-- --:--:-- 2123
100 57.3M  100 57.3M    0      0  95.4M      0 --:--:-- --:--:-- 95.4M
Client Version: v1.33.0
Kustomize Version: v5.6.0
ubuntu@ip-172-31-35-118:~$
```

Step 8.2: Setup Kubernetes Master Node

Launch a new Ubuntu 20.04 instance and run the following commands:



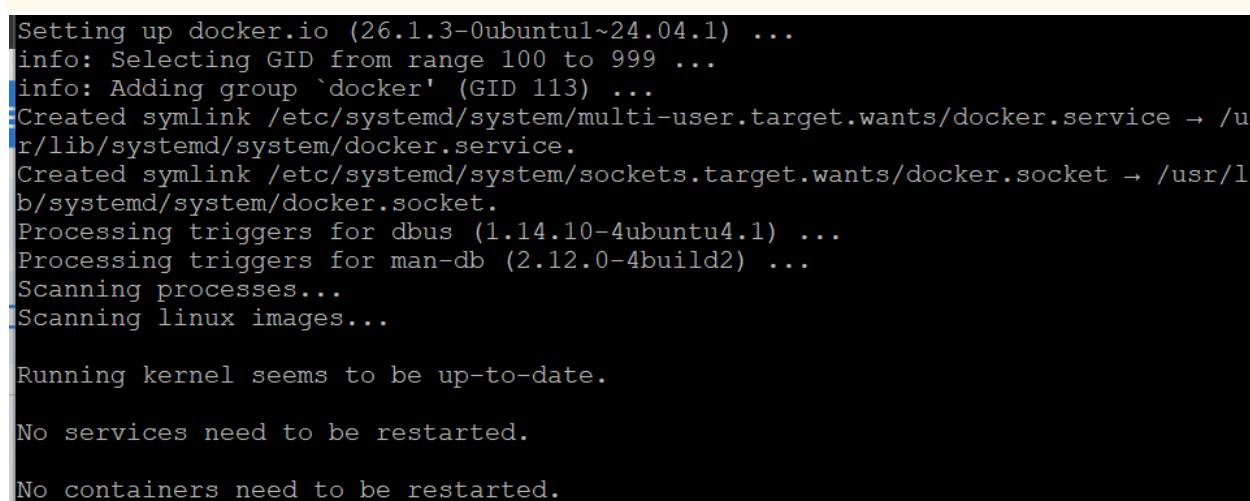
Name	Instance ID	Instance state	Instance type	Status check	Alarm status
master	i-0253be143b78286d0	Running	t2.medium	Initializing	View alarms +
worker	i-01a4203ecb780d71e	Running	t2.micro	Initializing	View alarms +
Jenkins-Dotnet	i-04a14bf06a286039	Running	t2.large	2/2 checks passed	View alarms +

```
sudo su
hostname master
bash
clear
```



```
root@master:/home/ubuntu
root@master:/home/ubuntu#
```

```
sudo apt-get update
sudo apt-get install -y docker.io
sudo usermod -aG docker Ubuntu
newgrp docker
sudo chmod 777 /var/run/docker.sock
```



```
Setting up docker.io (26.1.3-0ubuntu1~24.04.1) ...
info: Selecting GID from range 100 to 999 ...
info: Adding group `docker' (GID 113) ...
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /usr/lib/systemd/system/docker.service.
Created symlink /etc/systemd/system/sockets.target.wants/docker.socket → /usr/lib/systemd/system/docker.socket.
Processing triggers for dbus (1.14.10-4ubuntu4.1) ...
Processing triggers for man-db (2.12.0-4build2) ...
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.
```

```
# Install Kubernetes packages
sudo apt install -y apt-transport-https ca-certificates curl gpg

# Add Kubernetes GPG key
sudo mkdir -p /etc/apt/keyrings
curl -fsSL
https://pkgs.k8s.io/core:/stable:/v1.29/deb/Release.key | sudo
gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg

# Add Kubernetes repo
echo "deb
[signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] \
https://pkgs.k8s.io/core:/stable:/v1.29/deb/ /" | \
sudo tee /etc/apt/sources.list.d/kubernetes.list > /dev/null

# Update and install Kubernetes tools
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
```

```
Processing triggers for man-db (2.12.0-4build2) ...
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
root@master:/home/ubuntu# sudo apt-mark hold kubelet kubeadm kubectl
kubelet set on hold.
kubeadm set on hold.
kubectl set on hold.
root@master:/home/ubuntu# █
```

```
sudo kubeadm init --pod-network-cidr=10.244.0.0/16
```

To start using your cluster, you need to run the following as a regular user:

```
mkdir -p $HOME/.kube  
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config  
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Alternatively, if you are the root user, you can run:

```
export KUBECONFIG=/etc/kubernetes/admin.conf
```

You should now deploy a pod network to the cluster.

Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
<https://kubernetes.io/docs/concepts/cluster-administration/addons/>

Then you can join any number of worker nodes by running the following on each as root:

```
kubeadm join 172.31.16.125:6443 --token 6fu5zm.2r3bviuqups9nvgw \  
--discovery-token-ca-cert-hash sha256:55d72b504d2b77522c04c96a14fae837d4  
0fac3ce04e216b4fcfd4cc58fd0ec1d  
root@master:/home/ubuntu#
```

Note the `kubeadm join` command that will be displayed after initialization.
You'll need this for the worker node.

```
kubeadm join 172.31.29.105:6443 --token zun96o.fz68qzkq0mru8vb9  
\  
--discovery-token-ca-cert-hash  
sha256:4cb14662f40e4e4d423c3c4cf0a01ea33edd6a2b01836f0ce6a4f87cd  
f729f3f
```

```
# Exit from root and run as a regular user  
mkdir -p $HOME/.kube  
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config  
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Setup flannel

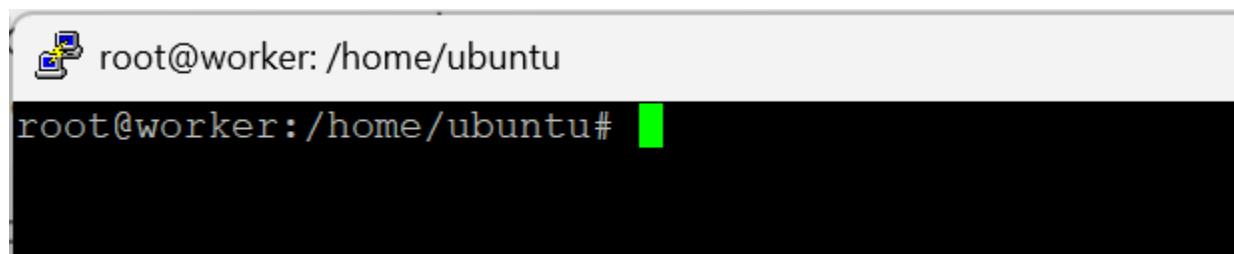
```
kubectl apply -f  
https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
```

```
ubuntu@ip-172-31-29-105:~$ mkdir -p $HOME/.kube
ubuntu@ip-172-31-29-105:~$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
ubuntu@ip-172-31-29-105:~$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
ubuntu@ip-172-31-29-105:~$ kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
namespace/kube-flannel created
clusterrole.rbac.authorization.k8s.io/flannel created
clusterrolebinding.rbac.authorization.k8s.io/flannel created
serviceaccount/flannel created
configmap/kube-flannel-cfg created
daemonset.apps/kube-flannel-ds created
ubuntu@ip-172-31-29-105:~$ █
```

Step 8.3: Setup Kubernetes Worker Node

Launch another Ubuntu 20.04 instance and run the following commands:

```
sudo su
hostname worker
bash
clear
```



A terminal window showing a root prompt on a worker node. The window title is 'root@worker: /home/ubuntu'. The command 'root@worker:/home/ubuntu#' is entered and followed by a green cursor bar.

Setup Docker

```
sudo apt-get update
sudo apt-get install -y docker.io
sudo usermod -aG docker Ubuntu
newgrp docker
sudo chmod 777 /var/run/docker.sock

Created symlink /etc/systemd/system/multi-user.target.wants/ubuntu-fan.service →
/usr/lib/systemd/system/ubuntu-fan.service.
Setting up docker.io (26.1.3-0ubuntu1~24.04.1) ...
info: Selecting GID from range 100 to 999 ...
info: Adding group `docker' (GID 113) ...
```

Setup Kubernetes

```
# Install Kubernetes packages
sudo apt install -y apt-transport-https ca-certificates curl gpg

# Add Kubernetes GPG key
sudo mkdir -p /etc/apt/keyrings
curl -fsSL
https://pkgs.k8s.io/core:/stable:/v1.29/deb/Release.key | sudo
gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg

# Add Kubernetes repo
echo "deb
[signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] \
https://pkgs.k8s.io/core:/stable:/v1.29/deb/ /" | \
sudo tee /etc/apt/sources.list.d/kubernetes.list > /dev/null

# Update and install Kubernetes tools
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
```

Use the join command from the master node output which has a structure like the following:

```
sudo kubeadm join <master-node-ip>:<master-node-port> --token
<token> --discovery-token-ca-cert-hash <hash>
```

```
kubeadm join 172.31.29.105:6443 --token zun96o.fz68qzkq0mru8vb9
\
--discovery-token-ca-cert-hash
sha256:4cb14662f40e4e4d423c3c4cf0a01ea33edd6a2b01836f0ce6a4f87cd
f729f3f
```

```

root@worker:/home/ubuntu# kubeadm join 172.31.29.105:6443 --token zun96o.fz68qzk
q0mr08vb9 \
    --discovery-token-ca-cert-hash sha256:4cb14662f40e4e4d423c3c4cf0a01ea33e
dd6a2b01836f0ce6a4f87cdf729f3f
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system g
et cm kubeconfig -o yaml'
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.y
aml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/ku
belet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.

root@worker:/home/ubuntu#

```

Step 8.4: Copy Kubernetes Config to Jenkins

Copy the Kubernetes config file from the master node to the Jenkins server:

On the Kubernetes master node:

```
cat $HOME/.kube/config
```

1. Copy the output.
2. Save this content on your local machine as `secret-file.txt`.

```

ubuntu@ip-172-31-29-105:~$ cat $HOME/.kube/config
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURCVEND
    QWUYZ0F3SUJBZ01JWXBGQTRtQUR4VDB3RFFZSkvWklodmNOQVFTEJRQXdgVEVUTUJFROExVUUKQXhN
    S2EzVmlaWEp1WhSbGN6QWVGdzB5T1RBムU1UUXdPVE0yTVRKYUZ3MHpOVEExTVRJd09UUxhNVGRhTUJV
    eApFekFSQmdOVkJBTVRDbXQxWW1WeWJtVjBaWE13Z2dFaU1BMEdDU3FHU01iM0RRRUJBUVVBQTRJQkR3
    QXdнZ0VLСkFvSUJBUNxaHzrakQrZnVvNFIweFZPTXYrWjZSQ3FEZkRvOTFycFdOaXFIMTdXQ3pqYmd6
    Vkh4V2VTTjVzYjUKV0JJcWU0V1k0dFBvT1pFVFVzUGRiaWRXdklrT2k4Wit2S010R2p1Z2xPdVgvQmlq
    ZHBDN2dvd1QvTUtZMy9nQgpyUzBpM1BybmZndWLSnQ5WFFTZSs3cEhQSWxkWlBSYXordnhMeFFEsm2
    dTFNWF1hUElhQm9RVFZyOVZwU1U3CkR1dUg2TVdkZUVFNEN5WitkMFhtSG5rWXN1TEkwcnZ3YXBLSVVN
    QnFLQn1OTzFWSFE3M1NQRVgxczY2a21nZ3YKbmRzb1Z1avFReTJhZH1hVGZMSit4NU16N1hqRGU5d05E
    d1R2aV1RZU9hdEQ1ZGRqUXNYZytUVU9jZTB4N1NpagpKbTJxaVVBeHM0SW9pYmJ4NzJRZVExd2NGMys3
    QWdNokFBR2DXVE.TYTUE0R0ExVWRFd0VCL3dRRUIF3SUNwREFOCkInT1ZTIIk1COWY4RUII0QURBUUQavTIIUw

```

```

dckr_pat_c8VSK63emct  ghp_d5b5qpCC4dc9Aig  sudo curl -fsL https://  secret-file.txt  trivy.txt  trivy-fs_report.txt
File Edit View
apiVersion: v1
clusters:
- cluster:
  certificate-authority-data:
LS0tLS1CRUdJTiBDRVJUSUZJQPFURSetLS0tCkIJSURCVENDQWUyZEF3SUJBz01JWXBGOTRtQU4RB3RFFZSktvNklodmNQVFTEJROXGdVEVUTUJFR0E8xUUUKQXHnS2EzVm1wEp1w1hSpGNGQIVGdzB5
T1RBMU1UUxDPVEoYTVRKYUZ3MhpOVEEXTRJd09UxhNVGrhTUJVbApFeKF5Qnd0VKJBTVRDbxQxW1WeWJtvjbaWE13Z2dFaU1BMeDU3FHu01iM0RRRJBUVVbQTRJQkR3QXdnZ0VLCKFvSUJBUUNxaHzr
akQrZnVNFIweFZPTXYrWjZSQ3FEZKrv0TFycfdoaxFIMtDQ3pqYm6Vkh4V2TTjvzjyJKv08JjcUUV1k0dFvT1pFvVzUGR1aWRxdKlT2k4Wit2s10r2p1z2xpdvgvQm1qzHBDN2dvd1qyTuTzMy9n
QgpyUzBpm1bybmZndVLSnQ5WFETZSs3cEnQSwxxLBsyXordnhMeXFESmd2dTfNwF1hElhQm9RvFzYovZvwlU13CkR1dug2TvdkZUVFNE5WitkMF5rWx11Ekwnz3YBLSVNNfLQn10tZFuSF3
M1N0RVgxcz2a21nZ3YkbmRzb1z1aVRetTjhZhlhVGZMS1t4NU16NhqRGU5d05Ed1r2aV1RZU9hdeQ1zGrqXNyzytUvU9jzTB4N1pagpk0TJxaVBElhM0sW9pym34NzJrzExd2Ngly3QwdnQkFBR2pX
VEJYJUE0R0ExwREd0VCL3dRRUf3SUnwREFQcKJnTzIU1kCQWY4RUJUQRBUUgVTU1wR0ExwREz1FxQkJUuHY3bjJ0TXYSeGZR1160VdtVkr8Ttx1w1tUQVVK0mdoVkhSRUVeakfNZ2dwcmRXSmxjbTVs
ZFdle1BMEdu3Fh101iM0RRRUIjDd1VR0TR10kFRQU1V1YUjNGvsagora99Pbh0tQDdxTfFzdk11S14bk47100bfY5TE1PM1z0nXh5211MfcvNxN0vW6MD14SXvTeDpC1z1hz18oc1p8RUS7RwZarz1k
ci93ZDYxcjvJMytzdm4ywHFVc1T01vekZneHjzTw1EYtD1S1prK2Uyc0UrU3pCL25uRvcKdnM0b1c2K3UxZzY1b0p0Q1h2dFhvYnFxUh6cko30EzrdDNCMzh2KytMRDcrR1g2k0NES18VMndIa1FWMTB2
cgpvd3Mybk15z1yG45K016cEfuaeXN5uKNIk0Q0aU8xYld1l00qxW1nSzVDS1h1zFN5eVjvRoRhoaY0VdzrJ3VXcmhLTUY31Foc2zrTucVyk4xS1V4YXv1U0042DN3MkJrcwVlQLV1420J0tmjsbEVYzmNn
OUYdwu9XHVLYw3VmKNNzIN1N2empCT0N3c10tLS0tRUEIENFU1RJRK1dQVRFLS0tLS0k
server: https://172.31.29.105:6443
name: kubernetes

```

Step 8.5: Install Kubernetes Plugin in Jenkins

1. Go to Jenkins → Manage Jenkins → Plugins → Available plugins
2. Search for "Kubernetes" and install the plugin. Also add kubernetes credentials

The screenshot shows the Jenkins Manage Jenkins interface under the Plugins section. The 'Download progress' tab is selected. On the left, there's a sidebar with links for Updates, Available plugins, Installed plugins, Advanced settings, and Download progress (which is highlighted). On the right, the 'Preparation' section lists three steps: Checking internet connectivity, Checking update center connectivity, and Success. Below that, the 'Pipeline: REST API' step is shown as 'Success'. The 'Pipeline: Stage View' and 'Loading plugin extensions' steps are also listed as 'Success'. Under the 'Kubernetes' section, 'Kubernetes Client API', 'Kubernetes Credentials', and 'Kubernetes' are all marked as 'Success'. The 'Loading plugin extensions' step is also marked as 'Success'.

Preparation	
• Checking internet connectivity	Success
• Checking update center connectivity	Success
• Success	

Pipeline: REST API	Success
Pipeline: Stage View	Success
Loading plugin extensions	Success

Kubernetes Client API	Success
Kubernetes Credentials	Success
Kubernetes	Success

Loading plugin extensions	Success
---------------------------	---------

3. Add Kubernetes credentials:

- Go to Jenkins → Manage Jenkins → Credentials → Add Credentials
- Kind: Secret file
- ID: kubernetes
- File: Upload the **secret-file.txt** file we saved earlier
- Click on Create

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

New credentials

Kind

Secret file

Scope ?

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

File

Choose file secret-file.txt

ID ?

kubernetes

Jenkins

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

Global credentials (unrestricted)

+ Add Credentials

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

ID	Name	Kind	Description
sonar-token	sonar-token	Secret text	sonar-token
docker	suhaimdvi/*****	Username with password	
kubernetes	secret-file.txt	Secret file	

Icon: S M L

Step 9: Update Jenkins Pipeline for Kubernetes Deployment

1. Create a `deployment.yaml` file in the K8S directory of your git repository:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: dotnet-app-deployment
spec:
  replicas: 1 # Set the desired number of replicas
  selector:
    matchLabels:
```

```

    app: dotnet-app
template:
  metadata:
    labels:
      app: dotnet-app
spec:
  containers:
    - name: dotnet-app
      image: suhaibmdv/dotnet-monitoring:latest # Replace
with your Docker image name
      ports:
        - containerPort: 5000 # Expose the port specified in
your Dockerfile

--- 

apiVersion: v1
kind: Service
metadata:
  name: dotnet-app-service
spec:
  selector:
    app: dotnet-app
  ports:
    - protocol: TCP
      port: 80 # Service port
      targetPort: 5000 # Port in the container
  type: NodePort # Use NodePort service type

```

2. Update the Jenkins pipeline to include Kubernetes deployment:

```

pipeline{
  agent any
  tools{
    jdk 'jdk17'
  }
  environment {

```

```
SCANNER_HOME=tool 'sonar-scanner'
}
stages {
    stage('clean workspace'){
        steps{
            cleanWs()
        }
    }
    stage('Checkout From Git'){
        steps{
            git branch: 'main', url:
'https://github.com/suhaib-md/Jenkins-CI-CD-.NET-WebApp-DevSecOp
s'
        }
    }
    stage("Sonarqube Analysis"){
        steps{
            withSonarQubeEnv('sonar-server') {
                sh ''' $SCANNER_HOME/bin/sonar-scanner
-Dsonar.projectName=Dotnet-Webapp \
-Dsonar.projectKey=Dotnet-Webapp '''
            }
        }
    }
    stage("quality gate"){
        steps {
            script {
                waitForQualityGate abortPipeline: false,
credentialsId: 'Sonar-token'
            }
        }
    }
    stage("TRIVY File scan"){
        steps{
            sh "trivy fs . > trivy-fs_report.txt"
        }
    }
}
```

```
        }
        stage("OWASP Dependency Check"){
            steps{
                dependencyCheck additionalArguments: '--scan ./ --format XML ', odcInstallation: 'OWASP-Dependency-Check'
                dependencyCheckPublisher pattern:
'**/dependency-check-report.xml'
            }
        }
        stage("Docker Build & tag"){
            steps{
                script{
                    withDockerRegistry(credentialsId: 'docker',
toolName: 'docker'){
                        sh "make image"
                    }
                }
            }
        }
        stage("TRIVY"){
            steps{
                sh "trivy image
suhaimdv/dotnet-monitoring:latest > trivy.txt"
            }
        }
        stage("Docker Push"){
            steps{
                script{
                    withDockerRegistry(credentialsId: 'docker',
toolName: 'docker'){
                        sh "make push"
                    }
                }
            }
        }
    }
    stage("Deploy to container"){

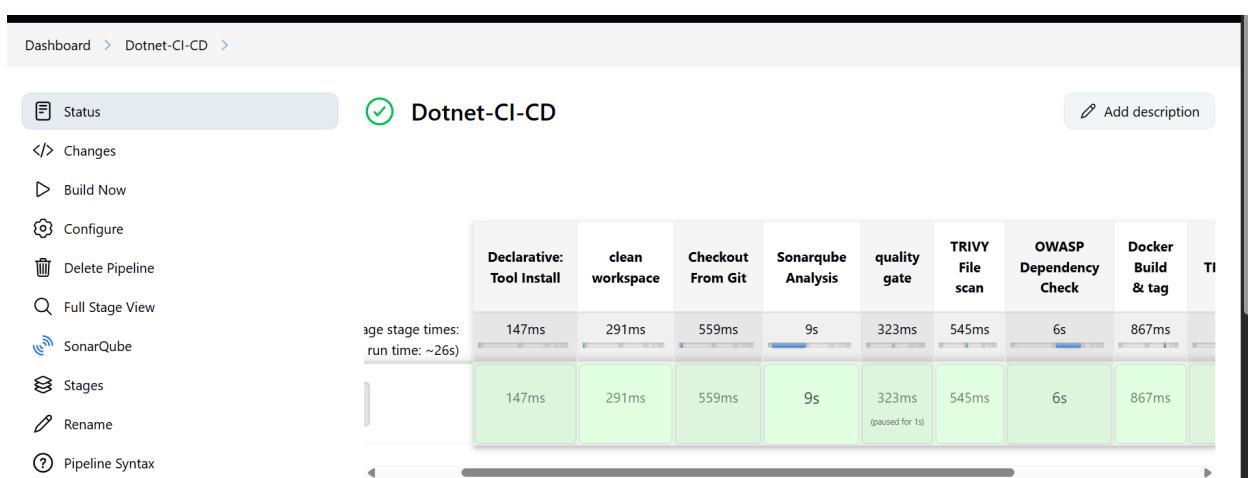

```

```

steps{
    sh "docker run -d --name dotnet -p 5000:5000
suhaibmdv/dotnet-monitoring:latest"
}
}
stage('Deploy to k8s'){
    steps{
        dir('K8S') {
            withKubeConfig(caCertificate: '', clusterName:
'', contextName: '', credentialsId: 'kubernetes', namespace: '',
restrictKubeConfigAccess: false, serverUrl: '') {
                sh 'kubectl apply -f deployment.yaml'
            }
        }
    }
}
}
}

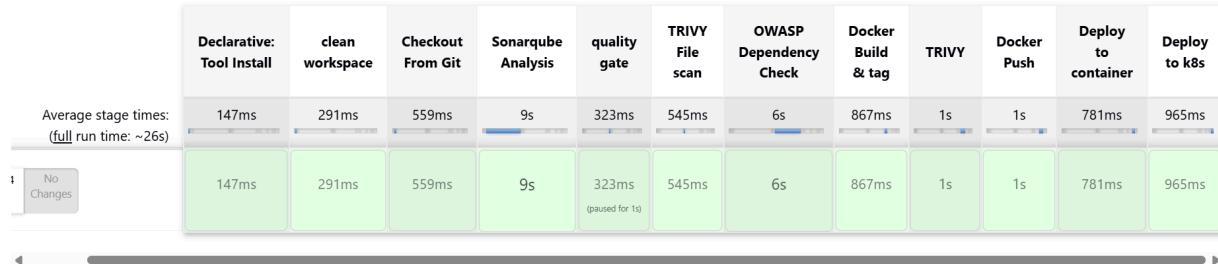
```

3. Click on Build Now and wait for the pipeline to complete executing



Dashboard > Dotnet-CI-CD > Full Stage View

t-CI-CD - Stage View



Dashboard > Dotnet-CI-CD > #12

```

[Pipeline] 
[Pipeline] sh
+ kubectl apply -f deployment.yaml
deployment.apps/dotnet-app-deployment created
service/dotnet-app-service created
[Pipeline] 
[kubernetes-cli] kubectl configuration cleaned up
[Pipeline] // withKubeConfig
[Pipeline] 
[Pipeline] // dir
[Pipeline] 
[Pipeline] // withEnv
[Pipeline] 
[Pipeline] // stage
[Pipeline] 
[Pipeline] // withEnv
[Pipeline] 
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS

```

We can look at the OWASP Despendency result at the dependency section

May 14 16:13 No Changes

147ms	291ms	559ms	9s	323ms (paused for 1s)	545ms	6s
-------	-------	-------	----	--------------------------	-------	----

SonarQube Quality Gate

Dotnet-Webapp Passed server-side processing: Success

Latest Dependency-Check

Permalinks

- Last build (#12), 14 min ago
- Last stable build (#12), 14 min ago
- Last successful build (#12), 14 min ago
- Last completed build (#12), 14 min ago

The Jenkins Dependency-Check Results dashboard shows a green status icon and a message indicating "No Vulnerabilities Found". The interface includes a sidebar with build-related links like Status, Changes, Console Output, and Git Build Data.

We can look at the Sonarqube Scan results at the Projects section of the Sonarqube dashboard

The Sonarqube Projects dashboard displays a single project named "Dotnet-Webapp" with a "Passed" status. The dashboard includes filters for Quality Gate (Passed, Failed), Reliability (A-E rating), and various code quality metrics like Bugs, Vulnerabilities, and Hotspots Reviewed.

The Sonarqube Project Information dashboard for "Dotnet-Webapp" shows a "Passed" quality gate status. It lists recent issues, including a minor bug in "src/Pages/Index.cshtml" with the description "Add an "alt" attribute to this image." and another in "src/Pages/Index.cshtml" with the same description.

Step 10: Access the Application

For Docker deployment:

`http://<Jenkins-EC2-Public-IP>:5000`

`http://18.233.115.206:5000/`

The screenshot shows a web browser window with the URL `18.233.115.206:5000` in the address bar. The page title is ".NET Demo Web App". The main content area contains the following text: "This is a modern .NET web app using the new minimal hosting model, and Razor pages. It has been designed with cloud demos & containers in mind, and for ease of deployment into Azure or Kubernetes. A simple app to use anytime you want something lightweight to run & deploy." Below this, a section titled "Basic features:" lists three items: "System status / information view", "Support for user sign-in with Azure AD and Graph API", and "App Insights support".

The screenshot shows a web browser window with the URL `18.233.115.206:5000/info` in the address bar. The page title is "System Information". The main content area is titled "Basic Details" and contains the following data:

Detail	Value
Containerized:	Looks like we're running in a Docker container! 😊
Kubernetes:	Not running in Kubernetes 🙄
App Insights:	Not enabled
Hostname:	198e3ba78590
OS Details:	Linux 6.8.0-1024-aws
Architecture:	X64 - 2 cores
Framework:	.NET 6.0.36

For Kubernetes deployment:

First get the service port using the following command in the master machine

```
kubectl get svc  
#copy service port  
<worker-ip:svc port>
```

```
ubuntu@master:~$ kubectl get svc  
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)  
AGE  
dotnet-app-service   NodePort   10.106.188.214 <none>        80:30259/TCP  
10m  
kubernetes       ClusterIP  10.96.0.1    <none>        443/TCP  
72m  
ubuntu@master:~$
```

<http://<Worker-Node-IP>:31000>
<http://18.234.162.37:30259/>

This is a modern .NET web app using the new minimal hosting model, and Razor pages. It has been designed with cloud demos & containers in mind, and for ease of deployment into Azure or Kubernetes. A simple app to use anytime you want something lightweight to run & deploy.

Basic features:

- System status / information view
- Support for user sign-in with Azure AD and Graph API
- App Insights support
- Geolocated weather info (from DarkSky API)
- Tools for triggering errors and high CPU load

Github Project

Container Images

.NET in Azure

The screenshot shows a web browser window with the URL `18.234.162.37:30259/info`. The page has a dark theme with purple navigation buttons. The main content area is titled "System Information". Under "Basic Details", there are several items:

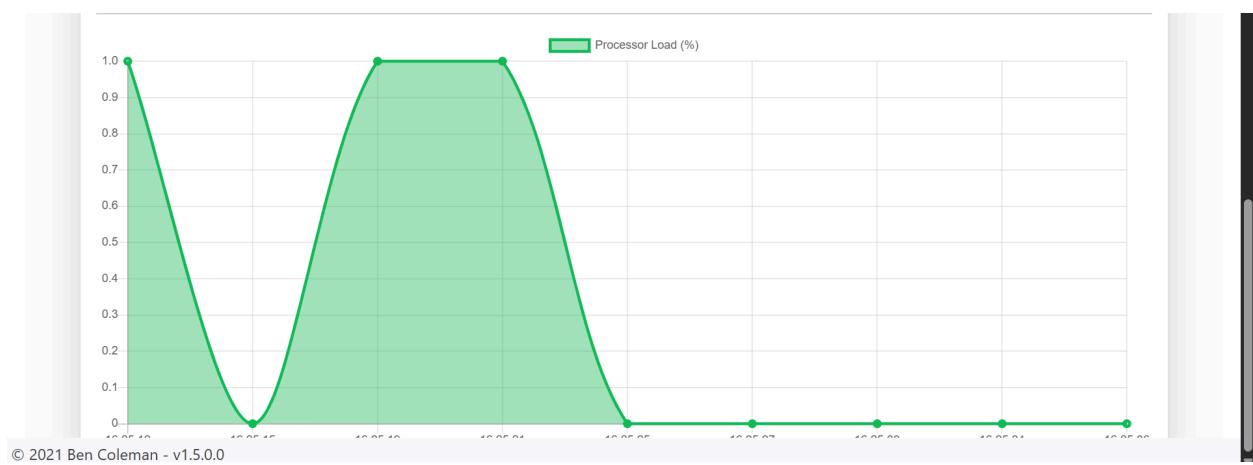
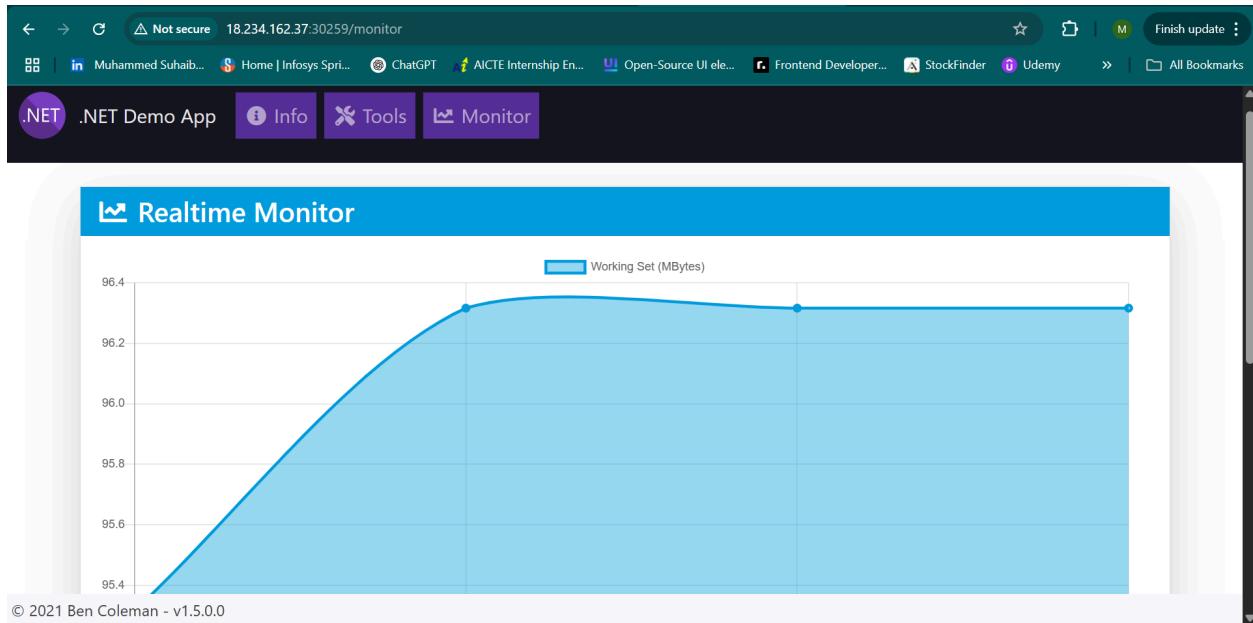
- Containerized: Looks like we're running in a Docker container! 😊
- Kubernetes: We're also running in a Kubernetes pod! 🤘
- App Insights: Not enabled
- Hostname: `dotnet-app-deployment-5d4b548fd7-v5g9c`
- OS Details: Linux 6.8.0-1024-aws
- Architecture: X64 - 1 cores
- Framework: .NET 6.0.36

At the bottom left, it says "© 2021 Ben Coleman - v1.5.0.0".

The screenshot shows a web browser window with the URL `18.234.162.37:30259/tools`. The page has a dark theme with purple navigation buttons. The main content area is titled "Tools". It lists five options:

- CPU Load: Generate CPU load, running iterations
- Alloc Memory: Force allocation of Mbytes of memory
- Exception: Error page, will trigger exception
- Garbage Collect: Force garbage collector to run
- 404 Page: Trigger a 404

At the bottom left, it says "© 2021 Ben Coleman - v1.5.0.0".



For Trivy Results:

Go to the successful pipeline build page and select workspaces from the left hand side menu

The screenshot shows a Jenkins pipeline build page for 'Dotnet-CI-CD' build #12. The sidebar on the left has several options: Console Output, Edit Build Information, Delete build '#12', Timings, Git Build Data, Dependency-Check, Pipeline Overview, Restart from Stage, Replay, Pipeline Steps, and Workspaces. The 'Workspaces' option is highlighted with a light gray background. In the main content area, it shows the build was started by user 'suhaiib' 24 minutes ago, took 26 seconds, and spent 14 ms waiting, 26 sec building, and 26 sec total from scheduled to completion. It also shows the git revision and repository information, and a note that there were no changes.

Click the link to view the workspace files

The screenshot shows the 'Workspaces for Dotnet-CI-CD #12' page. The sidebar on the left is identical to the previous screenshot. The main content area is titled 'Workspaces for Dotnet-CI-CD #12' and shows a single item: a link to '/var/lib/jenkins/workspace/Dotnet-CI-CD on built-in'. This indicates where the Trivy scan results are stored.

You can view all the files here. Trivy stores its scan results in this workspace.

The screenshot shows the Jenkins workspace interface. The left sidebar has links for Status, Console Output, and Workspace (which is selected). The main area is titled "Workspace" and shows a file tree. The "Workspace" folder contains several files and subfolders: CONTRIBUTING.md, dependency-check-report.xml, LICENSE, makefile, README.md, trivy.txt, and trivy-fs_report.txt. Below the file tree, there's a link to download all files as a zip and a note about hidden Tmp directories.

```

CONTRIBUTING.md      14 May 2025, 10:43:02   2.73 KiB
dependency-check-report.xml 14 May 2025, 10:43:20  13.60 KiB
LICENSE              14 May 2025, 10:43:02   1.04 KiB
makefile              14 May 2025, 10:43:02   2.63 KiB
README.md             14 May 2025, 10:43:02   9.46 KiB
trivy.txt             14 May 2025, 10:43:23  17.91 KiB
trivy-fs_report.txt  14 May 2025, 10:43:14    610 B

```

(all files in zip)

⚠ Tmp directories are hidden

Jenkins 2.504.1

To view the trivy scan results, download the folder as zip, extract and view as text file.

The terminal window shows the command used to run Trivy: `trivy -f json --severity critical ./app/dotnet-demoapp.deps.json` and the resulting JSON output. The output is a table showing vulnerabilities found in the target Docker image. The legend indicates that a dash '-' means 'Not scanned' and a zero '0' means 'Clean (no security findings detected)'.

Target	Type	Vulnerabilities	Secrets
suhaiibmdv/dotnet-monitoring:latest (alpine 3.20.5)	alpine	6	-
app/dotnet-demoapp.deps.json	dotnet-core	7	-
usr/share/dotnet/shared/Microsoft.AspNetCore.App/6.0.36/Microsoft.AspNetCore.Ap-p.deps.json	dotnet-core	0	-
usr/share/dotnet/shared/Microsoft.NETCore.App/6.0.36/Microsoft.NETCore.App.deps-.json	dotnet-core	0	-

Legend:
- '-': Not scanned
- '0': Clean (no security findings detected)

For OSS Maintainers: VEX Notice

If you're an OSS maintainer and Trivy has detected vulnerabilities in your project that you believe are not actually exploitable, consider issuing a VEX (Vulnerability Exploitability eXchange) statement.

Step 11: Clean Up

When done with the project, terminate all AWS EC2 instances to avoid unnecessary charges.

Troubleshooting

Error in Public Key

```
The following signatures couldn't be verified because the public key is not available:  
NO_PUBKEY 843C48A565F8F04B  
Reading package lists... Done  
W: GPG error: https://packages.adoptium.net/artifactory/deb noble InRelease: The following signatures couldn't be verified because the public key is not available: NO_PUBKEY 843C48A565F8F04B  
E: The repository 'https://packages.adoptium.net/artifactory/deb noble InRelease' is not signed.  
N: Updating from such a repository can't be done securely, and is therefore disabled by default.  
N: See apt-secure(8) manpage for repository creation and user configuration details.  
Reading package lists... Done  
Building dependency tree... Done  
Reading state information... Done  
E: Unable to locate package temurin-17-jdk  
.jenkins.sh: line 8: /usr/bin/java: No such file or directory  
deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian-stable binary/  
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
```

Changed to the current public key and GPG

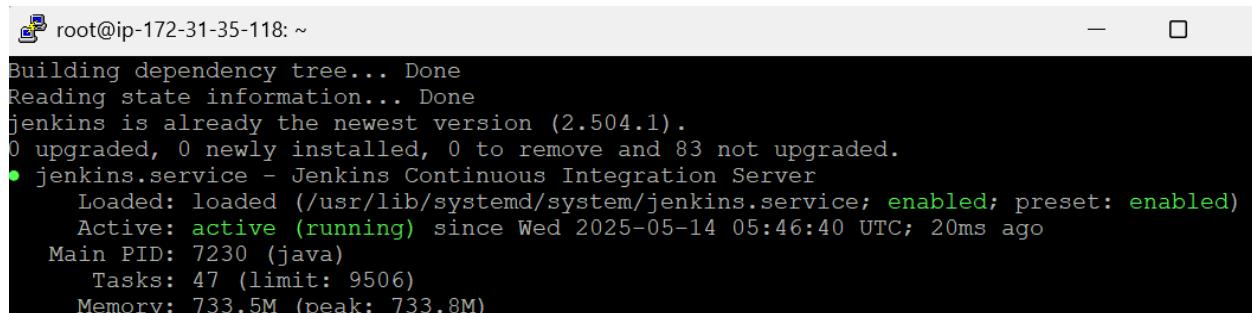
The versions which worked:

```
#!/bin/bash  
sudo apt update -y  
wget -O -  
https://packages.adoptium.net/artifactory/api/gpg/key/public |  
tee /etc/apt/keyrings/adoptium.asc  
echo "deb [signed-by=/etc/apt/keyrings/adoptium.asc]  
https://packages.adoptium.net/artifactory/deb $(awk -F=  
'/^VERSION_CODENAME/{print$2}' /etc/os-release) main" | tee  
/etc/apt/sources.list.d/adoptium.list  
sudo apt update -y  
sudo apt install temurin-17-jdk -y  
/usr/bin/java --version  
curl -fsSL
```

```

https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo
tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]
https://pkg.jenkins.io/debian-stable binary/ | sudo tee
/etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update -y
sudo apt-get install jenkins -y
sudo systemctl start jenkins
sudo systemctl status jenkins
#install docker
sudo apt-get update
sudo apt-get install docker.io -y
sudo usermod -aG docker ubuntu
newgrp docker
sudo chmod 777 /var/run/docker.sock
docker run -d --name sonar -p 9000:9000 sonarqube:lts-community
# install trivy
sudo apt-get install wget apt-transport-https gnupg lsb-release
-y
wget -qO -
https://aquasecurity.github.io/trivy-repo/deb/public.key | gpg
--dearmor | sudo tee /usr/share/keyrings/trivy.gpg > /dev/null
echo "deb [signed-by=/usr/share/keyrings/trivy.gpg]
https://aquasecurity.github.io/trivy-repo/deb $(lsb_release -sc)
main" | sudo tee -a /etc/apt/sources.list.d/trivy.list
sudo apt-get update
sudo apt-get install trivy -y

```

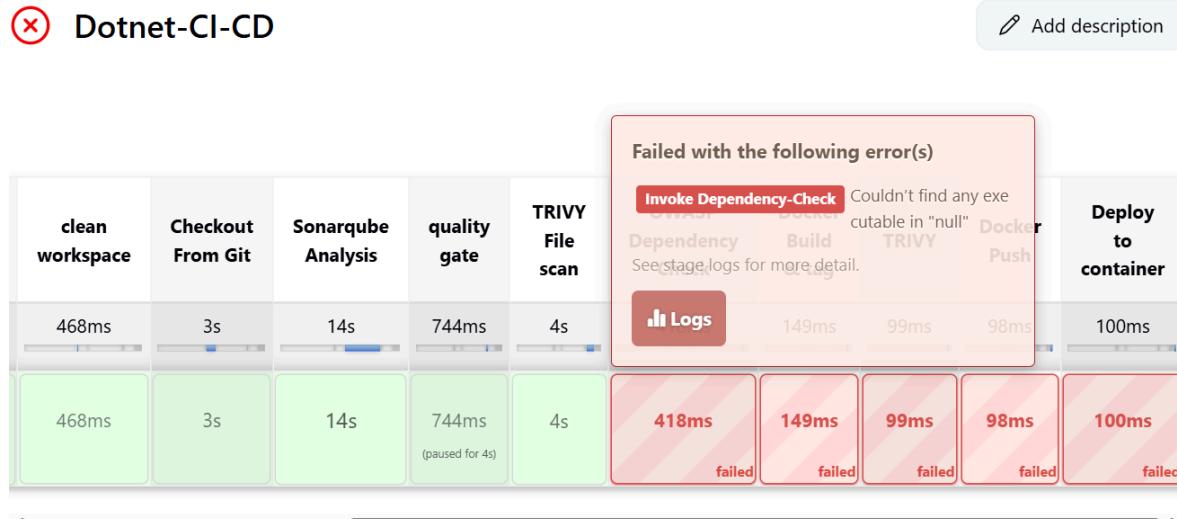


```

root@ip-172-31-35-118: ~
Building dependency tree... Done
Reading state information... Done
jenkins is already the newest version (2.504.1).
0 upgraded, 0 newly installed, 0 to remove and 83 not upgraded.
● jenkins.service - Jenkins Continuous Integration Server
    Loaded: loaded (/usr/lib/systemd/system/jenkins.service; enabled; preset: enabled)
      Active: active (running) since Wed 2025-05-14 05:46:40 UTC; 20ms ago
        Main PID: 7230 (java)
          Tasks: 47 (limit: 9506)
         Memory: 733.5M (peak: 733.8M)

```

Error while running pipeline



It is due to improper setup of owasp dependency check in jenkins tools.
Make sure you properly install it by selecting an installer.



Error while running pipeline (Docker)

Warning: failed to get default registry endpoint from daemon
(**Got** permission denied while trying to connect to the **Docker** daemon socket at unix:///var/run/docker.sock: **Get** http://%2Fvar%2Frun%2Fdocker.sock/v1.29/info: dial unix /var/run/docker.sock: connect: permission denied). Using system default: https://index.docker.io/v1/
Got permission denied while trying to connect to the **Docker** daemon socket at unix:///var/run/docker.sock: **Post** http://%2Fvar%2Frun%2Fdocker.sock/v1.29/auth: dial unix

```
/var/run/docker.sock: connect: permission denied
[Pipeline] // withDockerRegistry
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (TRIVY)
Stage "TRIVY" skipped due to earlier failure(s)
[Pipeline] getContext
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Docker Push)
Stage "Docker Push" skipped due to earlier failure(s)
[Pipeline] getContext
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Deploy to container)
Stage "Deploy to container" skipped due to earlier failure(s)
[Pipeline] getContext
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
ERROR: docker login failed
Finished: FAILURE
```

The error is:

```
Got permission denied while trying to connect to the Docker
daemon socket at unix:///var/run/docker.sock: Post
http://%2Fvar%2Frun%2Fdocker.sock/v1.29/auth: dial unix
/var/run/docker.sock: connect: permission denied
```

This is a common permission issue with Docker. The Jenkins user doesn't have the necessary permissions to access the Docker daemon socket. here's how to fix this:

Solution to the Docker permission denied error:

Add the Jenkins user to the Docker group:

```
sudo usermod -aG docker jenkins
```

Restart the Jenkins service to apply the changes:

```
sudo systemctl restart jenkins
```

Alternatively, you can change the permissions of the Docker socket (less secure approach):

```
sudo chmod 666 /var/run/docker.sock
```

```
ubuntu@ip-172-31-35-118:~$ sudo chmod 666 /var/run/docker.sock
ubuntu@ip-172-31-35-118:~$ █
```

Error while installing kubernetes

```
Hit:4 https://packages.cloud.google.com/apt kubernetes-xenial InRelease
Hit:5 http://security.ubuntu.com/ubuntu noble-security InRelease
Err:6 https://packages.cloud.google.com/apt kubernetes-xenial Release
  404  Not Found [IP: 142.250.31.139 443]
Reading package lists... Done
E: The repository 'https://apt.kubernetes.io kubernetes-xenial' does not have a Release file.
N: Updating from such a repository can't be done securely, and is therefore disabled by default.
N: See apt-secure(8) manpage for repository creation and user configuration details.
```

This error is because the correct keys are not used for our version of Ubuntu. To solve this the correct version is used.

```
# Install Kubernetes packages
sudo apt install -y apt-transport-https ca-certificates curl gpg

# Add Kubernetes GPG key
sudo mkdir -p /etc/apt/keyrings
curl -fsSL
https://pkgs.k8s.io/core:/stable:/v1.29/deb/Release.key | sudo
gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg

# Add Kubernetes repo
echo "deb
[signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] \
https://pkgs.k8s.io/core:/stable:/v1.29/deb/ /" | \
sudo tee /etc/apt/sources.list.d/kubernetes.list > /dev/null

# Update and install Kubernetes tools
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
```

GitHub Repository

The code for this project is available at:

<https://github.com/suhaib-md/Jenkins-CI-CD-.NET-WebApp-DevSecOps>

Conclusion

This DevSecOps project demonstrates how to set up a complete CI/CD pipeline for a .NET application using Jenkins, Docker, and Kubernetes. By following these steps, you've integrated security scanning with SonarQube, Trivy, and OWASP Dependency Check into the pipeline, ensuring that the application is not only built and deployed automatically but also checked for potential security vulnerabilities at various stages of the development lifecycle.

The pipeline shows the practical application of DevSecOps principles by:

1. Performing code quality and security analysis with SonarQube
2. Checking dependencies for vulnerabilities with OWASP Dependency Check
3. Scanning the file system and Docker images for vulnerabilities with Trivy
4. Automating the build, test, and deployment processes with Jenkins
5. Containerizing the application with Docker
6. Orchestrating containers with Kubernetes

This comprehensive approach helps ensure that security is an integral part of the development process rather than an afterthought.