



SIT223-SIT753
Credit Task:
**Continuous Integration and
DevSecOps in with Jenkins**

Continuous Integration and Deployment with Jenkins and GitHub

This is an individual Credit task.

Overview

In **Part 1** of this assessment task, you are required to create a Jenkins pipeline that integrates with GitHub. This section also provides hands-on experience with basic DevSecOps practices. **This part is mandatory for all students.**

Part 2 focuses on more advanced topics in DevOps, and you are required to **choose ONE of the following two options** based on your interest:

- 1) Extend your DevSecOps pipeline by integrating the SonarCloud.io service for code quality and security analysis.
- 2) Configure the Extended Email Notification Plugin in Jenkins to send custom build notifications via email.

You must select **ONLY ONE** of these options to complete the second part.

Part 1: Task 1 - GitHub Integration (**Mandatory for All Students**)

Task Instructions

1. Create a GitHub repository and configure it to be used with Jenkins.
2. Create a Jenkins job that will be responsible for running the pipeline. The job should be triggered automatically whenever a new commit is pushed to the GitHub repository. You **are NOT required** to configure a **webhook**; simply triggering the pipeline based on a time interval (e.g., using a scheduled poll for commits) will be sufficient for this part.
3. Define a pipeline with 7 stages. Each stage should have a specific task to perform. You will need to provide a description of the tasks performed in each stage and a tool that could be used.

Please note: This task is a theoretical exercise to design a pipeline. Unlike Task 2, you are not required to write or run a working script here—just describe the stages and tools. Only the specified tasks and tools should be printed.

The stages should include:

Stage 1: Build – Build the code using a build automation tool to compile and package your code. You need to specify at least one build automation tool, e.g., Maven.

Stage 2: Unit and Integration Tests – Run unit tests to ensure the code functions as expected and run integration tests to ensure the different components of the

application work together as expected. You need to specify test automation tools for this stage.

Stage 3: Code Analysis – Integrate a code analysis tool to analyse the code and ensure it meets industry standards. Research and select a tool to analyse your code using Jenkins

Stage 4: Security Scan – Perform a security scan on the code using a tool to identify any vulnerabilities. Research and select a tool to scan your code.

Stage 5: Deploy to Staging – Deploy the application to a staging server (e.g., AWS EC2 instance).

Stage 6: Integration Tests on Staging – Run integration tests on the staging environment to ensure the application functions as expected in a production-like environment.

Stage 7: Deploy to Production – Deploy the application to a production server (e.g., AWS EC2 instance).

Part 1: Task 2 - DevSecOps Basics (Mandatory for All Students)

In this task, you will use **npm** to run security tests on the [nodejs-goof](https://github.com/snyk-labs/nodejs-goof) project. This intentionally vulnerable Node.js application is designed for testing and demonstration purposes, making it an ideal sample for practicing security scanning. You will use **npm** to run security tests over the [nodejs-goof](https://github.com/snyk-labs/nodejs-goof) (<https://github.com/snyk-labs/nodejs-goof>) project.

Task Instruction

Step 1:

Create a GitHub repository named "8.2CDevSecOps". This will be the repository where you will clone the [nodejs-goof](https://github.com/snyk-labs/nodejs-goof) project. Then clone the [nodejs-goof](https://github.com/snyk-labs/nodejs-goof) project from its original GitHub repository:

```
git clone https://github.com/snyk-labs/nodejs-goof.git
cd nodejs-goof
git remote remove origin
git remote add origin https://github.com/your_github_username/8.2CDevSecOps.git
git push -u origin main
```

Once you have pushed all the project files into your repository, you can now write the pipeline script to run security tests via **npm**.

Please note: If you are using Docker, your Jenkins container may not have **Node.js** or **npm** installed. You can install **npm** inside your container. Please refer to Appendix I for details.

Step 2:

Create a pipeline and use the following scripts:

```
pipeline {
  agent any

  stages {
    stage('Checkout') {
      steps {
        git branch: 'main', url: 'https://github.com/your_github_username/8.2CDevSecOps.git'
      }
    }

    stage('Install Dependencies') {
      steps {
        sh 'npm install'
      }
    }

    stage('Run Tests') {
      steps {
        sh 'npm test || true' // Allows pipeline to continue despite test failures
      }
    }

    stage('Generate Coverage Report') {
      steps {
        // Ensure coverage report exists
        sh 'npm run coverage || true'
      }
    }

    stage('NPM Audit (Security Scan)') {
      steps {
        sh 'npm audit || true' // This will show known CVEs in the output
      }
    }
  }
}
```

Note: The example uses sh commands for Unix-based systems. If your Jenkins runs on Windows, replace sh with bat and adjust syntax (e.g., use `|| exit /b 0` instead of `|| true`).

You can use an online formatter tool to format your pipeline script (Groovy script):

<https://codebeautify.org/javaviewer/cbacc095>

Step 3:

Execute the pipeline and save the console output.

Part 2: Task 1 - Integrating SonarCloud.io into Your Pipeline

For this task, you will extend the pipeline you created in Part 1: Task 2 by adding a new 'SonarCloud Analysis' stage. The existing stages (Checkout, Install Dependencies, Run Tests,

Generate Coverage Report, and NPM Audit) should remain, as they prepare the codebase for SonarCloud analysis (e.g., generating coverage reports that SonarCloud can use). Your goal is to integrate SonarCloud.io into this workflow for advanced code quality and security checks.

Task Instruction

SonarCloud.io Setup

1. Set up your **SonarCloud.io** account and log in using your **GitHub account**. So, go to [SonarCloud.io](https://sonarcloud.io) and log in using your GitHub account.
2. Generate a **security token** by navigating to **Your Account > Security**. After creating the token, save it in a text file — you'll need it later when configuring Jenkins.
3. In SonarCloud, create a new **project**. During the project setup, connect it to your **GitHub repository** (which is named 8.2CDevSecOps in your GitHub account) that contains the nodejs-goof project.
4. Navigate to your local 8.2CDevSecOps repo folder (which should be the same as nodejs-goof, if you cloned the project there).

For example: `C:\Users\User-Name\nodejs-goof` if you followed the steps correctly.

5. Create the **sonar-project.properties** file in that folder.
6. Inside this file, add the following configuration (replacing placeholders with actual values).
 - **Project Key:** The **project key** is unique to your project in SonarCloud. To find it, go to the project overview page in SonarCloud. You'll see the project key in the URL as follows:

`https://sonarcloud.io/project/overview?id=<your_project_key>`

***For example:** if the URL is*

*`https://sonarcloud.io/project/overview?id=yourname_8.2CDevSecOps`, then your **project key** is `yourname_8.2CDevSecOps`.*

- **Organisation Name:** The organisation name refers to your SonarCloud organisation. To find it:
 - Navigate to **Your Account > Organisations** from the SonarCloud dashboard.
 - Your organisation name should be listed there. It's typically the name of the GitHub organisation or your personal GitHub username.
 - If your account is linked to a personal GitHub account, the organisation name will be your **GitHub username** (e.g., `yourname`).
- **Sonar Token:** please see Step 2 above.

```
# sonar-project.properties

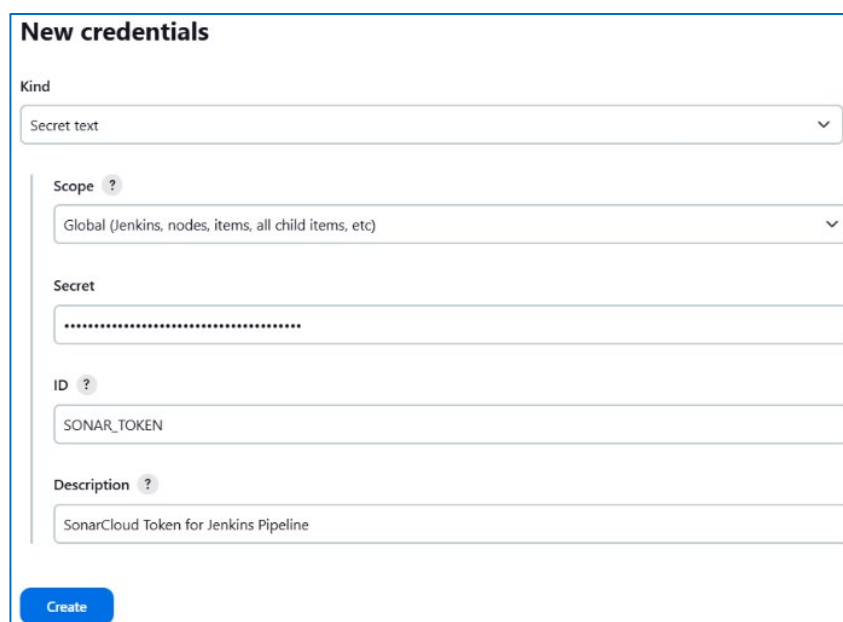
sonar.projectKey=<your_project_key>
sonar.organization=<your_organization_name>
sonar.host.url=https://sonarcloud.io
sonar.login=${SONAR_TOKEN}

sonar.sources=.
sonar.exclusions=node_modules/**,test/**
sonar.javascript.lcov.reportPaths=coverage/lcov.info

sonar.projectName=NodeJS Goof Vulnerable App
sonar.sourceEncoding=UTF-8
```

Jenkins Setup

- From the Jenkins dashboard, navigate to **Manage Jenkins > Credentials**.
 - Add a new credential using the **generated SonarCloud token** from the previous step.
 - Ensure you use **SONAR_TOKEN** as the token ID.
 - The credential type should be "Secret key".



- In your pipeline configuration, make sure to add this **SONAR_TOKEN** credential as an **environment variable**. This allows the pipeline to access the token securely during execution.

Hint: Use a `withCredentials` block or environment variable syntax in your pipeline to securely pass SONAR_TOKEN to the SonarScanner command.

- Add a new stage into your pipeline and name it "**SonarCloud Analysis**".

Note: If you are running Jenkins inside a Docker container, you may encounter issues downloading the **SonarScanner CLI** due to DNS resolution problems. Please refer to **Appendix II** for a possible solution.

4. **After executing the pipeline**, you should see **comprehensive security testing results** in the Jenkins console. These results are generated through **SonarCloud API calls**, which should be indicated by a **response code 200**. This means the SonarCloud analysis was successfully triggered and the results are being processed.
5. **Now, in your SonarCloud dashboard**, you should see a **detailed report** of your security scan. For example, it might highlight issues like a **hard-coded password in mongoose-db.js**. The dashboard will show findings such as vulnerabilities or security flaws detected by SonarCloud, providing you with the ability to inspect these issues further.

NodeJS Goof Vulnerable App

PUBLIC

Overview

Main Branch

Pull Requests

Branches

abkenar > NodeJS Goof Vulnerable App > main

SummaryIssuesSecurity HotspotsMeasuresCodeActivity

0.0% Security Hotspots Reviewed

To reviewFixedSafe

12 Security Hotspots to review

Review priority: High

Authentication

Review this potentially hard-coded password.

Review this potentially hard-coded password.

Review this potentially hard-coded password.

Review this potentially hard-coded password.

Command Injection

Review priority: Medium

Permission

Weak Cryptography

Review priority: Low

Others

12 of 12 shown

Review this potentially hard-coded password.

Hard-coded passwords are security-sensitive javascript:S2068

Status: To Review
This Security Hotspot needs to be reviewed to assess whether the code poses a risk.

Review

Review priority: High

Category: Authentication

Assignee: Not assigned

Where is the risk?

What's the risk?

Assess the risk

How can I fix it?

Activity

/mongoose-db.js

Show 46 more lines

47 User = mongoose.model('User');
48 User.find({ username: 'admin@nyk.io' }).exec(function (err, users) {
49 console.log(users);
50 if (users.length === 0) {
51 console.log('no admin');
52 new User({ username: 'admin@nyk.io', password: 'SuperSecretPassword' }).save(function (err, user, count) {

Review this potentially hard-coded password.

53 if (err) {
54 console.log('error saving admin user');
55 }
56 });
57 }
58 Show 1 more lines

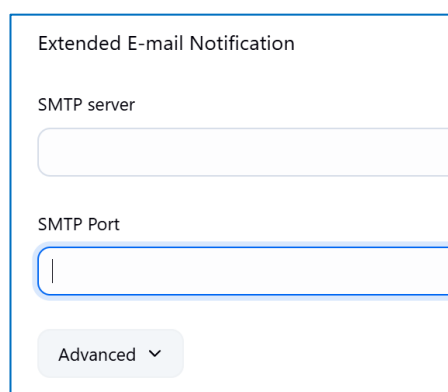
Part 2: Task 2 - Sending Email Notifications Using the Extended Email Plugin

In this part, you are required to integrate email notifications to inform developers about the build status and deployment events via an [email attachment](#). Additionally, for each stage of the pipeline, you will need to specify or research a tool that can be used for that particular stage.

Task Instruction

1. Configure the pipeline to send notification emails to a specified email address at the end of the **test** and **security scan** stages. The notification emails should include the status of the stage (success or failure) and [attach the logs](#).
2. Test the pipeline by making a few commits to the GitHub repository and ensuring that the pipeline runs successfully, and the notification emails are sent.

Hint: You are required to use the **Email Extension Plugin** (<https://plugins.jenkins.io/email-ext/>). Ensure that it is configured correctly first by going to **Manage Jenkins > System Configuration**.



Task Submission Instructions:

You must submit a **PDF file** containing **clickable** and **accessible links** (your tutor must be able to access the link) to the following short videos, in accordance with the Video Submission Guidelines:

Video Submission Guidelines

Video Quality

- Ensure that your video is clear and easy to view.
- Avoid recording your screen using a mobile phone, as this reduces clarity and professionalism.
- We highly recommend using **Panopto** or any reliable screen recording software.

Audio & Introduction:

- Your video must include sound.
- At the beginning of the video, clearly state your full name and student ID. This helps verify the authenticity of your submission.

Professional Presence:

- We encourage you to turn on your camera at the start or during your demo.
- As this is a unit on **Professional Practice**, presenting yourself professionally will strengthen your communication skills and overall presentation. This also helps verify the authenticity of your submission.

Sharing & Access:

- Before submitting your video link, ensure that your tutor has access to view it (eg, set appropriate sharing permissions on platforms like Panopto, YouTube, or OneDrive).

- **Part 1 – Task 1 (GitHub Integration with Jenkins – Mandatory for All Students)**

A short demo video (approximately 30–45 seconds) showcasing the successful execution of your Jenkins pipeline, including the console output. You must also demonstrate that the pipeline is triggered automatically after a new commit.

- **Part 1 – Task 2 (DevSecOps Basics – Mandatory for All Students)**

A short demo video (approximately 30–45 seconds) showcasing the vulnerability scan report by scrolling through the Jenkins console output.

You must submit one of the following:

- **Part 2 – Task 1 (DevSecOps with SonarCloud.io)**

A short demo video (approximately 1 minute) that demonstrates your pipeline script and the corresponding Jenkins console output. The video should also showcase your SonarCloud.io dashboard, highlighting the updates reflected after executing your pipeline.

- **Part 2 – Task 2 (Email Notification)**

A short demo video (approximately 1 minute) that clearly demonstrates your pipeline script, the successful execution of the pipeline, and the [email notification with log attachment that you receive at the end of the specified stages](#).

Appendix I – Installing npm on Jenkins Container

- 1) You must run your container with root access:

```
docker run -u root -p 8089:8080 -p 50000:50000 -v jenkins_home:/var/jenkins_home --name jenkins jenkins/jenkins:its
```

In this example, Jenkins has been installed on port 8089. Please make sure to specify the port number used for your own Jenkins installation. If you haven't changed the default configuration, the port should be 8080.

- 2) Connect to your container bash:

```
docker exec -it jenkins bash
```

In the above command, jenkins refers to the name of your Jenkins container. If the command doesn't work, ensure you're using the correct container name by running the `docker ps` command to list all active containers.

- 3) Once you are connected to the container in bash mode, you should see something similar to: `root@f849fcf32d3d:/#`

- 4) Now run the following commands:

```
apt update
apt install -y curl
curl -fsSL https://deb.nodesource.com/setup_18.x | bash -
apt install -y nodejs
```

Appendix II – Resolving DNS issue in Docker

Edit your Docker daemon config:

- Linux: `/etc/docker/daemon.json`
- Windows/Mac: Docker Desktop → Settings → Docker Engine

Add:

```
{  
  "dns": ["8.8.8.8", "8.8.4.4"]  
}
```

An updated daemon.json with DNS included would look like this:

```
{  
  "builder": {  
    "gc": {  
      "defaultKeepStorage": "20GB",  
      "enabled": true  
    }  
  },  
  "debug": false,  
  "experimental": false,  
  "insecure-registries": [],  
  "registry-mirrors": [],  
  "dns": ["8.8.8.8", "8.8.4.4"]  
}
```

Then restart Docker.