

SWINBURNE VIETNAM
HO CHI MINH CAMPUS
SUMMER 2024 SEMESTER



CLASS COS40006 – GROUP 2
Establishing a Smart Warehouse Management System

Instructor: Dr. Thomas Hang

Students:

Le Quang Hai – 104175779

Nguyen Tai Minh Huy – 104220352

Truong Duc Sang - 104220420

HO CHI MINH CITY – JUNE 2024

Contents

Introduction	4
Consultation Phase	4
Project description	4
Project objectives	5
Improve Inventory Accuracy:.....	5
Reduce Operational Costs:.....	6
Minimize Human Errors:.....	6
Streamline Operations:	7
Problem statement	8
Preliminary design concepts	10
Scrapped Ideas	10
Design and Development	10
Design Phase.....	10
Development Phase	11
Expected Outcomes	12
Compatibility of Design	12
Specifications	12
Explanation.....	13
Frontend.....	13
Backend.....	16
Data display	29
Product Sample.....	31
Project Outcomes	33
Improved Development Efficiency.....	33
Optimized Resource Utilization	34
Enhanced Monitoring and Maintenance	34

Streamlined Operations and Improved Collaboration..... 34

Business Impact..... 35

Conclusion 35

Introduction

By installing a cutting-edge Smart Warehouse Management System (WMS), Gemadept Corporation is starting a massive overhaul of its warehouse operations. To improve operational accuracy and efficiency, this program aims to incorporate cutting-edge technologies such as RFID technology, Internet of Things sensors, and data analytics. Gemadept Corporation is seeking proposals from seasoned technology suppliers to develop, deploy, and maintain this cutting-edge system in order to accomplish this.

Implementing a Smart WMS aims to lower operating expenses, limit human error, increase inventory accuracy, and optimize warehouse operations. Real-time tracking and monitoring streamlined procedures for data retrieval and storage, automated data collection, and improved decision-making via sophisticated analytics are among the anticipated results.

The Swinburne team recommends taking on this difficult project in order to create, set up, and manage Gemadept Corporation's Smart Warehouse Management System. The group will accomplish this by applying its expertise in cutting-edge technology solutions. Our comprehensive approach will ensure that data analytics, IoT sensors, and RFID technologies are seamlessly integrated to achieve the desired improvements in warehouse operations. Gemadept Corporation's warehouse operations are expected to become a model of efficacy and efficiency because of the Smart WMS deployment, which will optimize inventory management, reduce costs, minimize errors, and streamline procedures.

Consultation Phase

Project description

Gemadept Corporation has made a project to install a Smart Warehouse Management System (WMS) in order to transform its warehouse operations. This system will create an extremely accurate and efficient warehouse environment by utilizing RFID technology, cutting edge IoT sensors, and advanced data analytics.

Real-time tracking of items, automated data capture procedures, and enhanced visibility into warehouse operations—which can result in up to 80% more organizational efficiency—are all made possible by the Smart WMS. IoT sensors will follow the flow of goods and continuously check environmental conditions to ensure accurate records and ideal storage. RFID technology will expedite inventory counts, shipping, and receiving by enabling quick and accurate item identification.

Advanced data analytics will offer powerful insights into warehouse performance, helping to optimize operations and proactively address potential issues. The system will seamlessly integrate with other business systems, ensuring better coordination and informed decision-making across the organization.

The Smart WMS will also enhance order fulfillment processes, from picking to packing and shipping, ensuring accuracy and efficiency at every stage. This comprehensive approach will improve inventory management, speed up order fulfillment, and ultimately increase customer satisfaction.

Gemadep Corporation hopes to establish a smooth and streamlined workflow that improves overall operational efficiency by putting this Smart Warehouse Management System into place. This will set the company up for future growth and success in the logistics sector.

Project objectives

Improve Inventory Accuracy:

Integrate RFID technology and IoT tracking devices to achieve a 95% accuracy rate in inventory management. This entails placing IoT sensors throughout all storage and transit zones and tagging every inventory item with an RFID tag.

The implementation will result in:

- **Reduction in discrepancies:** Reduce inventory discrepancies by 90%, ensuring data accuracy and integrity.
- **Stockout prevention:** Decrease stockout incidents by 80%, maintaining optimal inventory levels to meet demand.
- **Efficiency improvements:** Improve inventory tracking and auditing efficiency by 70%, reducing manual labor and associated costs.

- **Real-time updates:** Enable real-time inventory level updates with 99% uptime, providing continuous visibility into stock status.

By leveraging these advanced technologies, we will streamline inventory processes, enhance visibility, and improve overall operational efficiency.

Reduce Operational Costs:

Decrease warehousing costs by 20% by optimizing storage and retrieval processes through the following measures:

- **Implement Automated Guided Vehicles (AGVs)** to automate the transportation of goods within the warehouse, reducing manual labor hours by 40%.
- Install **conveyor systems** in high-traffic areas to facilitate faster and more efficient movement of items, cutting transport time by 30%.
- **Utilize vertical lift modules (VLMs)** to capitalize on unused vertical space, increasing storage capacity by 30%.
- Introduce **mobile shelving and racking systems** to compact storage areas, improving space utilization by 25%.
- Upgrade to **LED lighting and implement motion sensors** to reduce energy consumption by 20%.
- Implement **smart climate control systems** to optimize heating and cooling, lowering utility costs by 15%.
- Establish a **preventive maintenance schedule** for equipment and infrastructure, reducing repair costs by 10%.
- Integrate **advanced warehouse management software (WMS)** to improve order picking accuracy by 99% and reduce inventory tracking errors by 50%, cutting associated costs by 20%.
- Utilize **real-time data analytics** to monitor inventory levels and predict demand, reducing overstock and understock situations by 25%.

These strategic actions will collectively lead to a 20% reduction in overall warehousing costs, enhancing operational efficiency, and boosting profitability.

Minimize Human Errors:

Reduce human errors in inventory handling by 30% through automated data capture and processing. This initiative includes the following actions:

- **Automated Data Capture:** Deploy RFID and barcode scanning systems to automatically capture inventory data during receipt, storage, and dispatch. This will eliminate manual data entry errors, improving accuracy by 35%.
- **Real-Time Inventory Tracking:** Implement IoT sensors and GPS tracking to provide real-time visibility of inventory movement and location, reducing the risk of misplaced items and ensuring accurate data recording.
- **Integration with Warehouse Management Systems (WMS):** Integrate automated data capture tools with advanced WMS to streamline data processing and synchronization, enhancing data integrity and reducing discrepancies by 25%.
- **Automated Reconciliation Processes:** Use automated reconciliation tools to compare inventory records against physical counts regularly, quickly identifying and correcting discrepancies to maintain accuracy.
- **Training and Standard Operating Procedures (SOPs):** Develop and implement comprehensive training programs and SOPs for staff to ensure correct usage of automated systems, further minimizing errors due to incorrect handling.

These measures will ensure that data is accurately recorded and processed, minimizing the risk of mistakes that typically arise from manual entry. The overall impact will be a 30% reduction in human errors in inventory handling, enhancing operational efficiency and accuracy.

Streamline Operations:

Achieve a 20% reduction in average order fulfillment time within six months by integrating advanced analytics and real-time data visibility into the Smart Warehouse Management System (WMS). This initiative includes the following actions:

- **Advanced Analytics:** Implement predictive analytics to forecast demand and optimize inventory levels, ensuring that popular items are always in stock and reducing delays caused by stockouts.
- **Real-Time Data Visibility:** Utilize IoT sensors and RFID technology to provide real-time visibility of inventory movement and location. This will enable faster decision-making and reduce the time spent searching for items.
- **Smart WMS Integration:** Integrate advanced analytics and real-time data visibility tools into the Smart WMS to streamline order processing workflows, automate picking and packing, and optimize route planning within the warehouse.

- **Automated Picking Systems:** Deploy automated picking systems, such as robotic pickers and conveyor belts, to speed up the order picking process and reduce manual handling time.
- **Employee Training and Optimization:** Provide training for warehouse staff on the new systems and processes to ensure efficient usage and maximize the benefits of the technology.
- **Continuous Monitoring and Improvement:** Establish continuous monitoring and feedback loops to identify bottlenecks and areas for improvement, enabling ongoing optimization of the fulfillment process.

These measures will enhance decision-making capabilities, speed up processing times, and improve overall operational efficiency. The overall impact will be a 20% reduction in average order fulfillment time within six months.

Problem statement

Several obstacles prevent Gemadep Corporation's warehousing operations from operating as accurately and efficiently as possible. An increased risk of human mistake, operational expenses, and inventory anomalies are caused by manual operations and antiquated systems. Stockouts, delays in order fulfillment, and inefficient utilization of warehouse space are the outcomes of these problems.

A complete and integrated solution is required to address these issues. This system makes use of contemporary technology to streamline warehouse operations, automate procedures, and give real-time visibility. The implementation of a Smart Warehouse Management System (WMS) aims to resolve these issues through the following measures:

- **Automation of Processes:** Integrate automated data capture tools, such as RFID and barcode scanners, to reduce manual data entry errors and enhance data accuracy. Implement automated picking systems, including robotic pickers and conveyor belts, to speed up order processing and reduce handling time.
- **Real-Time Visibility:** Utilize IoT sensors and GPS tracking to monitor inventory movement and location in real-time. This will provide accurate, up-to-date information on stock levels, reducing the risk of stockouts and improving order fulfillment times.
- **Advanced Analytics:** Implement predictive analytics to forecast demand and optimize inventory levels, ensuring popular items are always in stock and minimizing delays.

Use data analytics to identify and address inefficiencies in the warehouse layout and processes.

- **Cost Reduction:** Upgrade to energy-efficient lighting and climate control systems to lower utility costs. Streamline inventory management to reduce excess stock and associated carrying costs.
- **Error Minimization:** Establish automated reconciliation processes to regularly compare inventory records against physical counts, quickly identifying and correcting discrepancies. Develop comprehensive training programs for staff to ensure correct usage of new systems and reduce human error.
- **Operational Streamlining:** Optimize warehouse layout to maximize space utilization and improve the flow of goods. Integrate the Smart WMS with existing systems to create a seamless, end-to-end inventory management solution.

Preliminary design concepts

Scrapped Ideas

- **Fully Autonomous Robots Response:** We thought about using fully autonomous robots for all warehouse operations, but decided to scrap this idea due to high initial costs and the complexity of implementation. The return on investment just wasn't there for us at this time.
- **Blockchain for Inventory Tracking:** Using blockchain to track inventory was an intriguing idea. However, we scrapped it because the integration challenges were significant and the added value was minimal compared to simpler, more established solutions.
- **Drones for Internal Deliveries:** We considered using drones to transport items within the warehouse. This idea was scrapped due to safety concerns and potential regulatory hurdles that would complicate implementation.
- **VR for Warehouse Training:** Virtual reality training for staff sounded like a cutting-edge solution, but we decided to scrap it. It was deemed unnecessary and too costly for our current training needs and wouldn't provide enough additional benefit.

Design and Development

Design Phase

1. Assessment and Analysis
 - **Current State Analysis:** Conduct a comprehensive assessment of existing warehouse operations, identifying key pain points, inefficiencies, and areas for improvement.
 - **Requirements Gathering:** Engage with stakeholders to gather detailed requirements for the Smart WMS, ensuring alignment with business objectives.
2. Conceptual Design
 - **System Architecture:** Design the overall architecture of the Smart WMS, including the integration of RFID, IoT sensors, AGVs, conveyor systems, robotic pickers, and advanced analytics tools.
 - **Technology Selection:** Choose appropriate technologies and vendors for RFID systems, IoT sensors, automated systems, and WMS software.

3. Detailed Design

- **Data Capture and Integration:** Design the layout and configuration of RFID readers, barcode scanners, and IoT sensors to ensure comprehensive data capture and seamless integration with the WMS.
- **Process Automation:** Develop detailed designs for the deployment of AGVs, conveyor systems, and robotic picking systems, ensuring they integrate smoothly with existing processes.
- **User Interface:** Design user-friendly dashboards and interfaces for real-time data visibility and analytics, tailored to the needs of warehouse staff and management.
- **Warehouse Layout Optimization:** Create optimized layouts incorporating vertical lift modules (VLMs) and mobile shelving systems to maximize space utilization.

Development Phase

1. Prototyping and Testing

- **Pilot Implementation:** Implement a pilot version of the Smart WMS in a controlled section of the warehouse to test the integration of RFID, IoT sensors, AGVs, and conveyor systems.
- **User Feedback:** Gather feedback from warehouse staff and refine the system based on their input, addressing any issues or areas for improvement.

2. Full-Scale Implementation

- **RFID and Barcode Systems:** Tag all inventory items with RFID or barcode labels and install fixed RFID readers and handheld barcode scanners.
- **IoT Sensors:** Deploy IoT sensors to monitor environmental conditions and track inventory locations in real-time.
- **Smart WMS Integration:** Fully integrate the Smart WMS with existing ERP and supply chain systems, configuring dashboards and analytics tools for real-time data visibility.
- **Process Automation:** Deploy AGVs, conveyor systems, and robotic pickers to automate the transportation and picking of goods.

3. Training and SOP Development

- **Staff Training:** Develop and deliver comprehensive training programs for warehouse staff on the use of new technologies and systems.
- **Standard Operating Procedures (SOPs):** Create and distribute detailed SOPs to ensure consistent and correct handling of inventory and usage of systems.

4. Optimization and Continuous Improvement

- **Monitoring and Feedback:** Establish continuous monitoring of system performance and collect feedback from staff to identify areas for improvement.
- **Data Analysis:** Use real-time data analytics to identify bottlenecks and optimize processes.
- **Continuous Improvement:** Implement ongoing improvements to enhance efficiency and effectiveness of warehouse operations.

Expected Outcomes

- 20% reduction in average order fulfillment time within six months.
- 30% reduction in human errors in inventory handling.
- 20% decrease in warehousing costs through improved space utilization, automation, and energy efficiency.
- Enhanced real-time visibility and decision-making capabilities, leading to better inventory management and customer satisfaction

Compatibility of Design

Specifications

The design of our DevOps project emphasizes compatibility across various systems and environments to ensure seamless integration, deployment, and operation. The key specifications include:

- **Cross-Platform Compatibility:** Our design supports multiple operating systems, including Windows, Linux, and macOS, enabling developers to work in their preferred environment.
- **Containerization:** Utilizing Docker ensures that our application can run consistently across different environments, from development to production.
- **CI/CD Integration:** We integrate continuous integration and continuous deployment tools (GitHub Actions and Jenkins) to automate the build, test, and deployment processes.
- **Cloud Compatibility:** Our solution is designed to work seamlessly with cloud services such as AWS, ensuring scalability and reliability.
- **Security:** Incorporates security scanning tools like OWASP and Trivi to ensure the application is free from vulnerabilities.

- **Monitoring and Logging:** Implementing Prometheus and Grafana for monitoring and logging ensures real-time insights into system performance and health.

Explanation

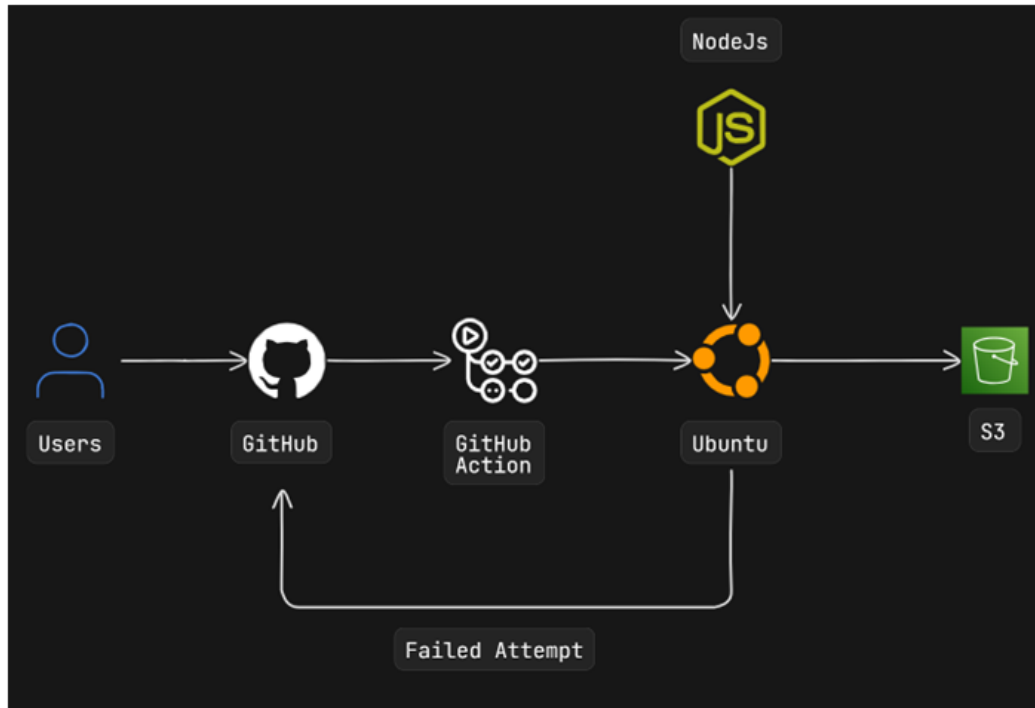
Frontend

Objectives & Technology stack

We anticipated that the front end would receive frequent updates, particularly in terms of UI changes. Therefore, a solution that is simple and cost-effective to update is essential for our application. Additionally, quick loading speed is crucial, making static web hosting an ideal choice for this scenario. Below are the technologies used in the CI/CD workflow for this application, along with the reasons for their selection:

- **GitHub Workflow:** Utilized for streamlined version control and continuous integration, ensuring automated and consistent deployment processes.
- **Amazon S3 (Static Web Hosting):** Chosen for its scalability, reliability, and cost-effectiveness in delivering fast and efficient static content.
- **NodeJS (Application Environment):** Selected for its robust performance in server-side scripting and its extensive ecosystem of libraries and tools.
- **Ubuntu (Build Environment):** Preferred for its stability, security, and compatibility with various CI/CD tools and frameworks.

Workflow



The CI/CD workflow is triggered when a user make a commit on main, which is the production branch of the front end, which trigger the GitHub Action:

```
name: Build and upload website to S3

on:
  push:
    branches:
      - master
```

Code Snippet 1.1: Event Triggered

The next stage of the CI/CD Workflow is initialize an environment, install node and front end dependencies:

```
jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v2

      - name: Set up Node.js
        uses: actions/setup-node@v2
        with:
          node-version: '>=20.0.0'

      - name: Install dependencies
        run: yarn install

      - name: Build the project
        run: yarn build
```

Code Snippet 1.2: Environment setup & Dependency installation

The final stage of the front-end deployment is setting up AWS Credentials using environment variables and upload build files to AWS S3:

```
- name: Configure AWS Credentials
  uses: aws-actions/configure-aws-credentials@v1
```

```
with:
  aws-access-key-id: ${ secrets.ACCESS_KEY_AWS }
  aws-secret-access-key: ${ secrets.SECRET_KEY_AWS }
  aws-region: us-east-1

- name: Deploy static site to S3 bucket
  run: aws s3 sync ./dist/ s3://swe3003-koala-react-fe --delete
```

Code Snippet 1.3: AWS Credential setup & File upload

Backend

Workflow Objectives & Technology stack

The backend server is designed to uphold best practices in code maintenance, secure deployment and development platforms, and a scalable production environment. System access may vary widely—some users will have extensive access while others may have none—making flexibility in scaling up and down crucial. Additionally, to manage complex workflows, robust monitoring tools are essential. To meet these criteria, the workflow must include quality checks, security checks, dependency checks, and platform checks for security vulnerabilities, as well as a containerized architecture for seamless scalability. The following tools are employed to achieve these goals:

- **Jenkins:** For automating the build and deployment processes, ensuring consistent and reliable CI/CD pipelines.
- **SonarQube:** For continuous inspection of code quality and security, detecting bugs and vulnerabilities.
- **OWASP Scan:** For identifying security flaws in the code, based on OWASP guidelines.
- **Trivy:** For comprehensive vulnerability scanning of containers.
- **Docker:** For containerizing applications to enable consistent environments across development and production.
- **Kubernetes:** For orchestrating containerized applications, ensuring efficient scaling and management.
- **Prometheus:** For monitoring and alerting, providing detailed metrics on application performance.
- **Grafana:** For visualizing data from Prometheus and Node Exporter, creating intuitive dashboards.

- **Node Exporter:** For exporting hardware and OS metrics exposed by *NIX kernels for Prometheus monitoring

AWS Cloud Infrastructure

To run the mentioned workflow with necessary tools, an AWS infrastructure must be setup to met the requirements, which outlined as follows:

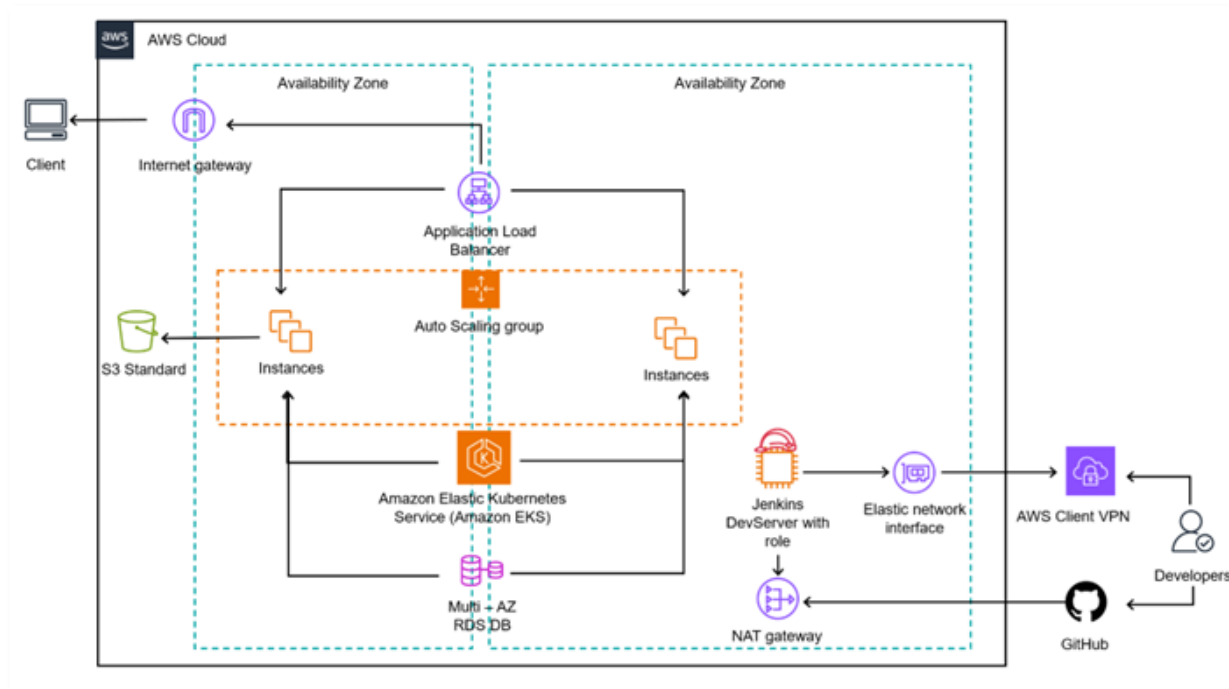


Figure 2: Cloud Infrastructure

In this abstract version of the cloud infrastructure, subnet configuration and security groups are not detailed in the diagram. The selected services and their usage for the backend are listed below:

- **AWS EC2:** Hosts the Jenkins development server.
- **Amazon EKS:** Manages Kubernetes instances.
- **Multi-AZ RDS:** Hosts the database.
- **Load Balancer:** Balances load between nodes.
- **Elastic Network Interface:** Allows developers to access Jenkins using VPN.
- **AWS Client VPN:** Enables developers to access resources in Jenkins within a private subnet.
- **NAT Gateway:** Allows Jenkins to access the internet, such as the GitHub code repository.
- **AWS S3:** Object storage to store files created in run time.

- **IAM Role:** Give Jenkins the credentials for environment variables used in the Kubernetes instances.

Resource allocation concentration

The resources are divided into three focus areas: Client-focused, Platform-focused, and Developer-focused. Detailed descriptions of each are provided below:

Client-focused

This part of the architecture is exposed to the public internet and includes:

- **Application Load Balancer:** Faces the internet gateway, allowing requests to reach the backend Kubernetes instances.
- **AWS S3:** Store and serve file created during backend code execution

Platform-focused

This part of the architecture serves the internet-facing components but operates in a network-isolated environment. It includes:

- **Auto-scaled Kubernetes Instances:** Ensures efficient scaling and management.
- **Amazon EKS:** Manages Kubernetes clusters.
- **Database (Multi-AZ RDS):** Provides reliable and scalable database hosting.

Developer-focused

This part of the architecture operates in a private subnet but is accessible via VPN. It includes:

- **Jenkins Development Server:** Runs in a network-isolated environment for enhanced security.
- **Elastic Network Interface:** Provides secure access for developers using VPN.
- **AWS Client VPN:** Ensures private access to Jenkins resources.
- **NAT Gateway:** Facilitates Jenkins' access to external resources, such as GitHub.
- **IAM Role:** Provide credentials for Jenkins to create env files for the application.

Workflow

Below is the demonstration of the workflow used in this project:

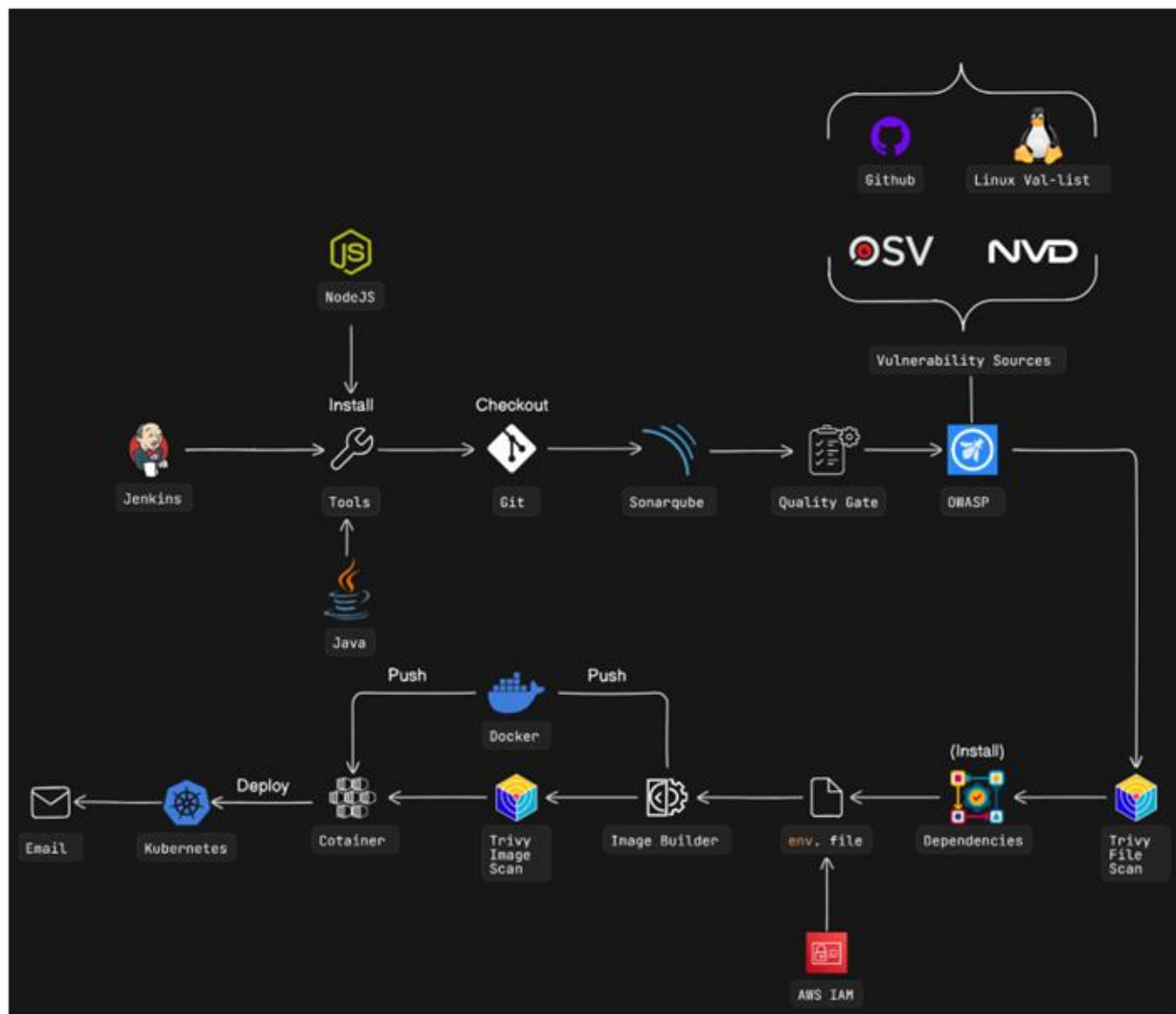


Figure 3: Workflow Overview

The Jenkins pipeline is configured as follows:

- **Agent:** The pipeline runs on any available agent.
- **Tools:** Specifies the tools needed:
 - JDK 17
 - Latest Node.js
- **Environment Variables:**
 - SCANNER_HOME is set to the location of the SonarQube scanner.

The first stage ensures a clean working environment by performing the following actions:

- **Clean Workspace:** Removes all files from the workspace.
- **Stop and Remove Docker Container:** Attempts to stop and remove any running Docker containers named koala-restaurant to prevent conflicts.

```
stage('clean workspace'){
    steps{
        cleanWs()
        script {
            try {
                sh "docker stop management-system"
            } catch (Exception e) {
                echo "Error stopping docker container:
${e.getMessage()}"
            }
            try {
                sh "docker rm management-system"
            } catch (Exception e) {
                echo "Error removing docker container:
${e.getMessage()}"
            }
        }
    }
}
```

Code Snippet 2.1: Clean Workspace Stage

The next stage is checkout from the GitHub repository and branch:

```
stage('Checkout from Git'){
    steps{
        git branch: 'main', url: 'https://github.com/mahou-anisphia/management-system.git'
    }
}
```

Code Snippet 2.2: Git Checkout

The next stage of the workflow will run a code analysis using SonarQube to ensure code's security and quality standards are met:

```
stage('SonarQube Analysis') {
    steps {
        withSonarQubeEnv('sonar-server') {
            sh ''' $SCANNER_HOME/bin/sonar-scanner -
Dsonar.projectName=Management_System \
-Dsonar.projectKey=Management_System '''
        }
    }
}
```

Code Snippet 2.3: SonarQube Analysis

As checking for code's quality is finished, the next stage of the workflow is it waits for the quality check result from SonarQube and fail the deployment if the criteria are not met:

```
stage('Quality Gate') {
    steps {
        script {
            try {
                def qg = waitForQualityGate abortPipeline:
false, credentialsId: 'Sonar-token'
                if (qg.status != 'OK') {
                    error "Pipeline aborted due to quality
gate failure: ${qg.status}"
                }
            } catch (Exception e) {
                echo "Quality gate check failed with error:
${e.message}"
            }
        }
    }
}
```

Code Snippet 2.4: Quality Gate stage

The next stage will run a project dependencies check scan using OWASP:

```
stage('OWASP FS SCAN') {
    steps {
```

```
        dependencyCheck additionalArguments: '--scan ./ --  
disableYarnAudit --disableNodeAudit', odcInstallation: 'DP-Check'  
        dependencyCheckPublisher pattern: '**/dependency-  
check-report.xml'  
    }  
}
```

Code Snippet 2.5: OWASP Scan Check

Another scan, focusing on project files using Trivy:

```
stage('TRIVY FS SCAN') {  
    steps {  
        sh "trivy fs . > trivyfs.txt"  
    }  
}
```

Code Snippet 2.6: Trivi Scan Check

The next stage is focused on the dependency installation for the application using the npm package manager:

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

Code Snippet 2.7: Dependency Installation

The next stage in the pipeline create an env file for the application, which contain the credentials in the IAM role. This credential allow the application to access AWS Services.

```
stage('Create .env File') {  
    steps {  
        writeFile file: '.env', text: '''\  
DB_ENDPOINT=database-1.c30qgks8ysva.us-east-1.rds.amazonaws.com  
DB_USER=anisphia  
DB_PASSWORD=magicology  
DB_NAME=restaurant_system  
JWT_TOKEN=management_token
```

```
ACCESS_KEY_AWS=AKIA4WJZ4T2M3PGQYGUG
SECRET_KEY_AWS=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
DEFAULT_REGION_AWS=us-east-1
S3_BUCKET_NAME=management-system-swe-unit-bucket
CONNECTION_LIMIT=10
'''
    }
}
```

Code Snippet 2.8: ENV Created

After all the needed files are created and configured, the next stage will build the docker image, and push into Docker Hub, ready to deploy:

```
stage("Docker Build & Push"){
    steps{
        script{
            withDockerRegistry(credentialsId: 'docker',
toolName: 'docker'){
                sh """
                    docker build -t management-system .
                """
                sh "docker tag koala-restaurant
anisphia/management-system:latest"
                sh "docker push anisphia/management-
system:latest"
            }
        }
    }
}
```

Code Snippet 2.9: Docker Build & Push

As the build process may raise some security vulnerabilities, the docker image is then scanned by Trivy before deployment:

```
stage("TRIVY"){
    steps{
        sh "trivy image anisphia/management-system:latest >
trivyimage.txt"
```

```
}  
}
```

Code Snippet 2.10: Trivi Image Scan

After all security scans, the docker image is deployed in the local development environment first to enable easy interaction for developers and testers:

```
stage('Deploy to container') {  
    steps {  
        script {  
  
            sh 'docker run -d --name management-system -p  
8081:3000 anisphia/management-system:latest'  
  
        }  
    }  
}
```

Code Snippet 2.11: Docker Deploy Local

Finally, the container is deployed in Kubernetes for production environment:

```
stage('Deploy to kubernetes'){  
    steps{  
        script{  
            dir('Kubernetes') {  
                withKubeConfig(caCertificate: '', clusterName:  
'', contextName: '', credentialsId: 'k8s', namespace: '',  
restrictKubeConfigAccess: false, serverUrl: '') {  
                    sh 'kubectl apply -f deployment.yml'  
                    sh 'kubectl apply -f service.yml'  
                }  
            }  
        }  
    }  
}
```

Code Snippet 2.12: Kubernetes deployment

Before the workflow is over, the final stage will notify developers through email on their deployment status and the logs to detect and fix anything if the deployment is failed:

```
post {  
  always {  
    emailx attachLog: true,  
          subject: "'${currentBuild.result}'",  
          body: "Project: ${env.JOB_NAME}<br/>" +  
                "Build Number: ${env.BUILD_NUMBER}<br/>" +  
                "URL: ${env.BUILD_URL}<br/>",  
          to: 'ducsangtruong2004@gmail.com',  
          attachmentsPattern: 'trivyfs.txt,trivyimage.txt'  
    }  
  }  
}
```

Code Snippet 2.13: Email Notification & Logs

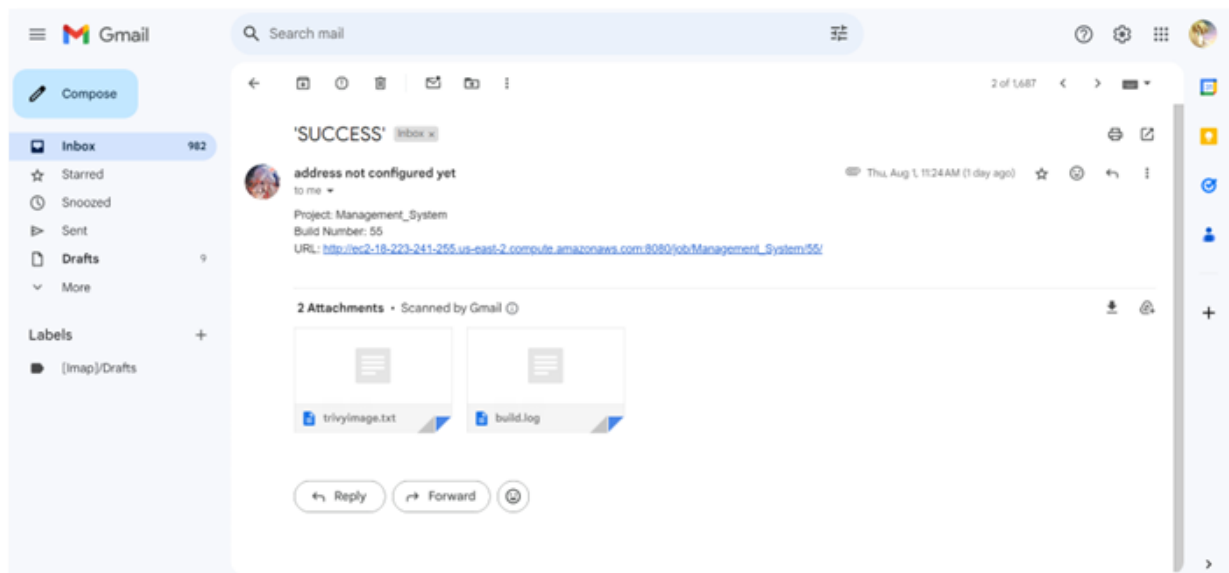


Figure 4: Auto sent emails

Quality Assurance & System Monitoring

As mentioned earlier, as the workflow for the back end is complex, and there are multiple return values and statistics of each process (i.e. Security check, quality check, dependency check, image vulnerability check, etc), a monitoring system is necessary. There are 2 main

criteria to be measured in this workflow, one is the criteria generated by analyzing the code, the other is the criteria monitoring the system's attributes.

Code analysis insights

In the first criteria, focusing on code's quality and security, we used SonarQube, Trivy and OWASP Scan to gather insights and analytics. SonarQube provide quality and security rating of the code, as shown below:

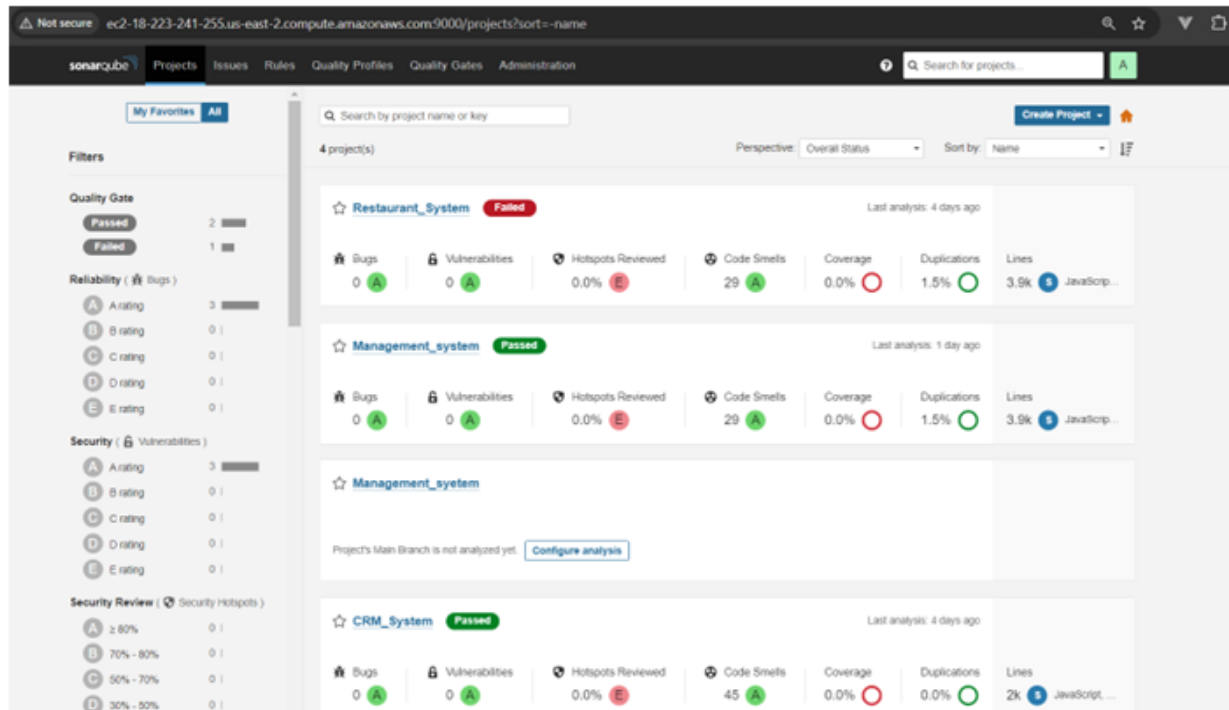


Figure 5: SonarQube Code Rating

Additionally, a deep analysis of issue in code and time estimated to fix them is provided:

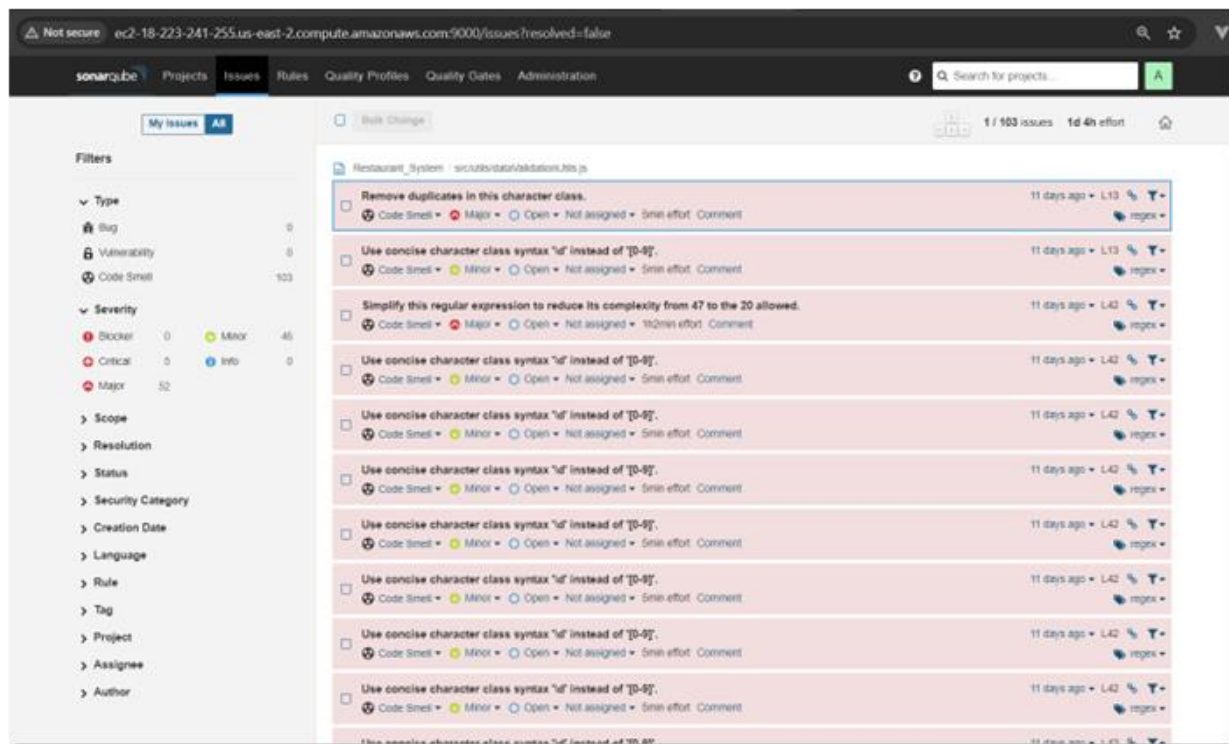


Figure 6: Code issue and time estimated for fixing

SonarQube also provide security gate, allowing developers to view the result directly in Jenkins:

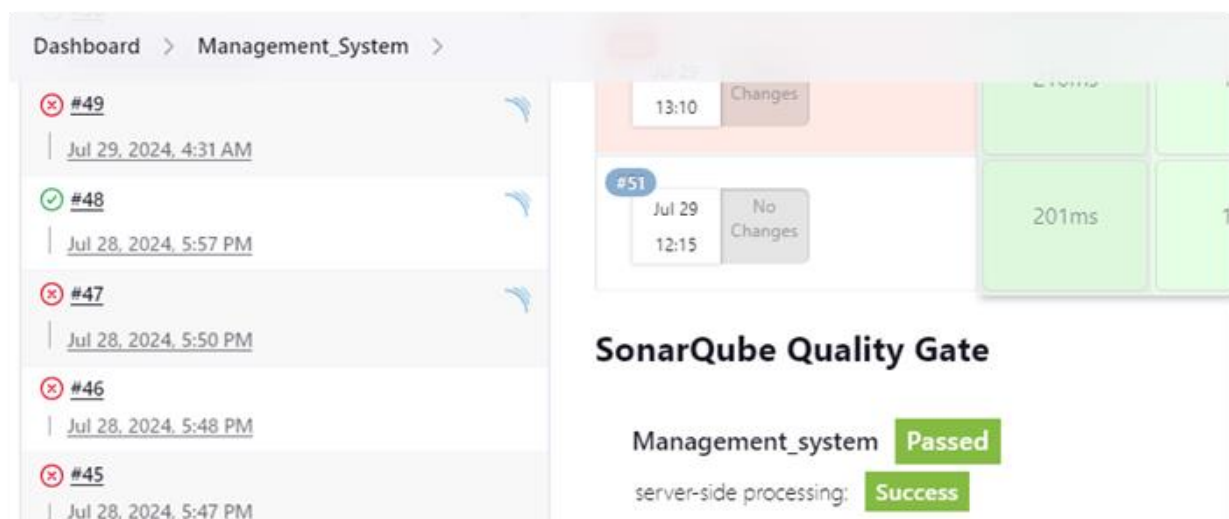


Figure 7: SonarQube Quality Gate Result

The next stage of the quality check will use OWASP Dependency check, which produce the result as follows:

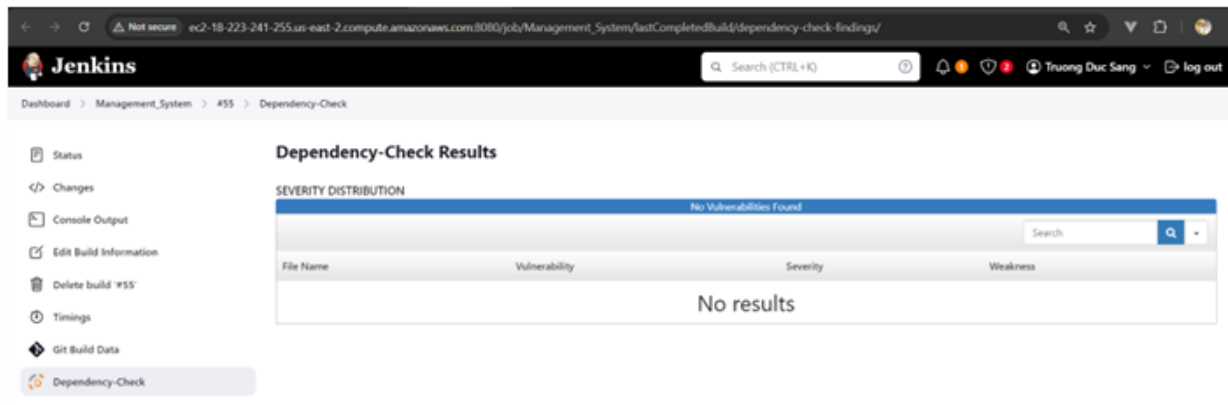


Figure 8: OWASP Check Result

The final stage of the check focused on the file system and any potential vulnerabilities in the image itself using Trivy, which produce a chart as follows:

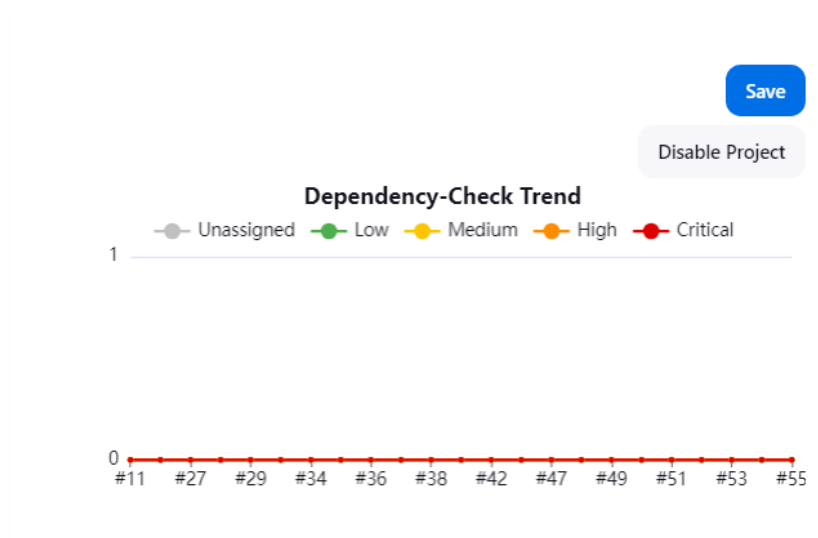


Figure 9: Trivi Check Result

System monitoring

To allow developers get a better understanding of how the server is operating, a set of system monitor is configured in the Jenkins instance, which mostly take input from Node explorer:

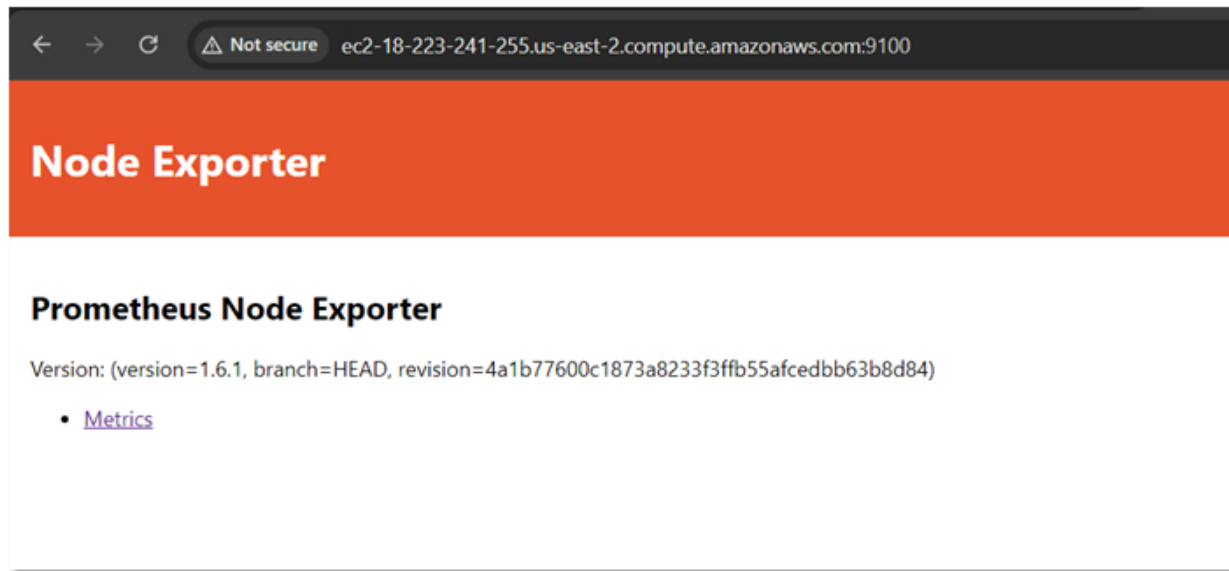


Figure 10: Node Explorer running on port 9100

Data display

Our project provides a user-friendly interface for monitoring and managing the deployment pipeline. Key data display features include:

- **Dashboard:** Real-time display of CI/CD pipeline status, including current builds, tests, and deployments.
- **Log Viewer:** Detailed logs from various stages of the pipeline for debugging and monitoring purposes.
- **Metrics:** Visual representation of key performance indicators (KPIs) such as build times, deployment success rates, and resource usage.
- **Alerts:** Notifications and alerts for pipeline failures, security vulnerabilities, and performance issues.

To gather all data collected from node explorer and services running on the server, we've configured Prometheus to listen to the data source:

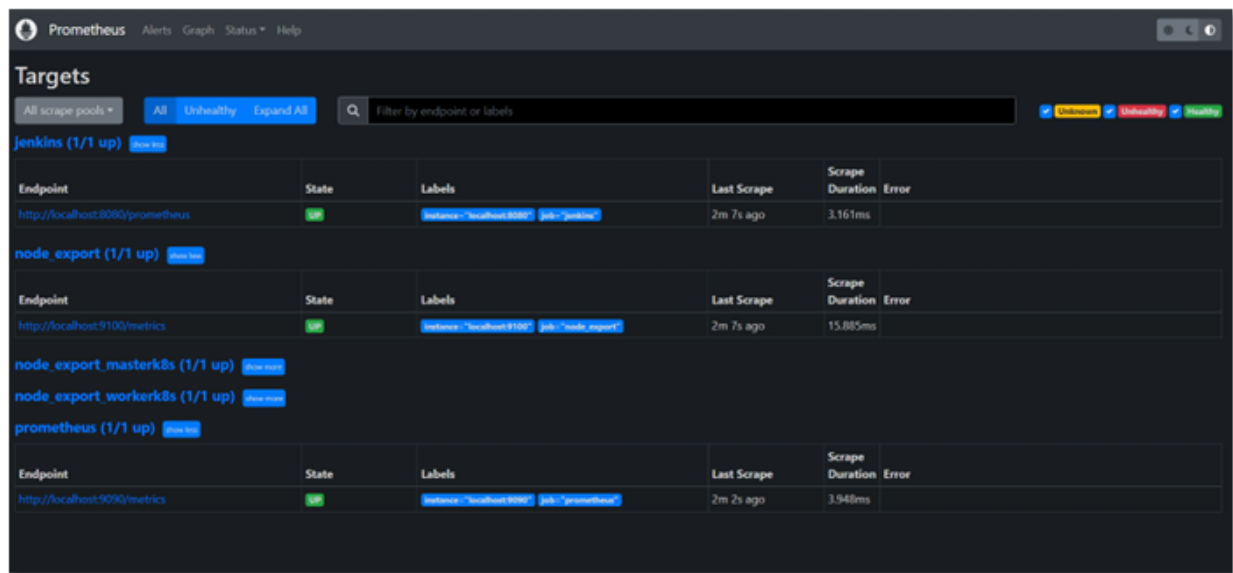


Figure 11: Promotheus Server Monitor

Finally, to visualize the data, allowing developers to get quick insights, we run Grafana as the data visualization tool, showing build analysis and system status:

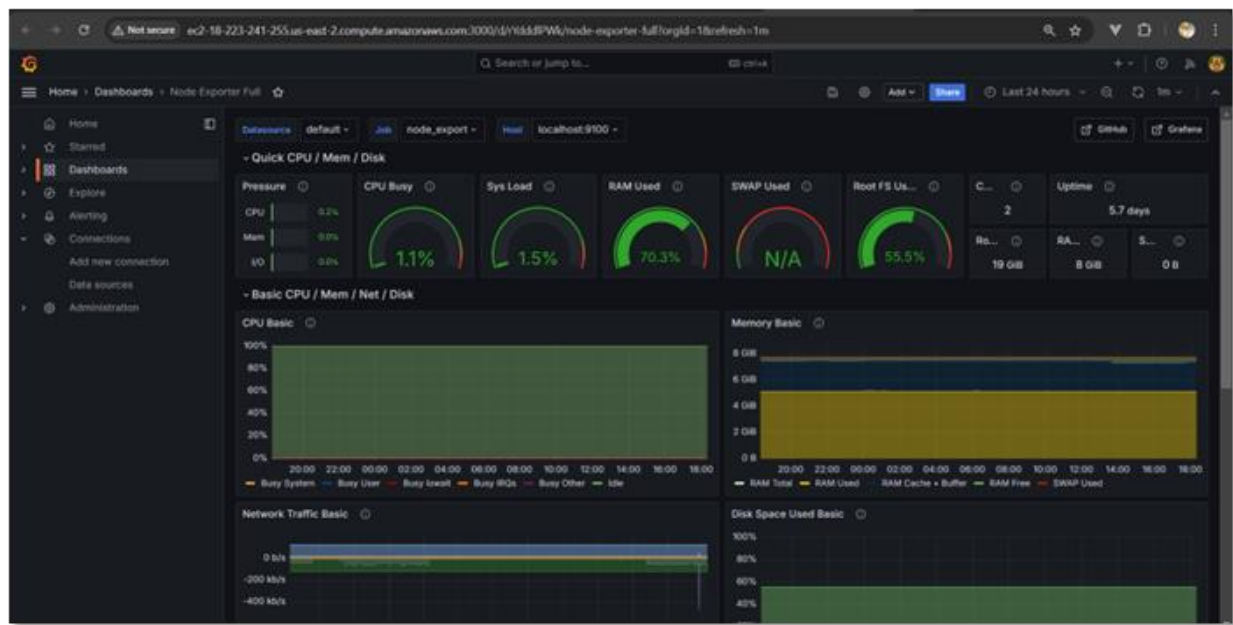


Figure 12: System Status

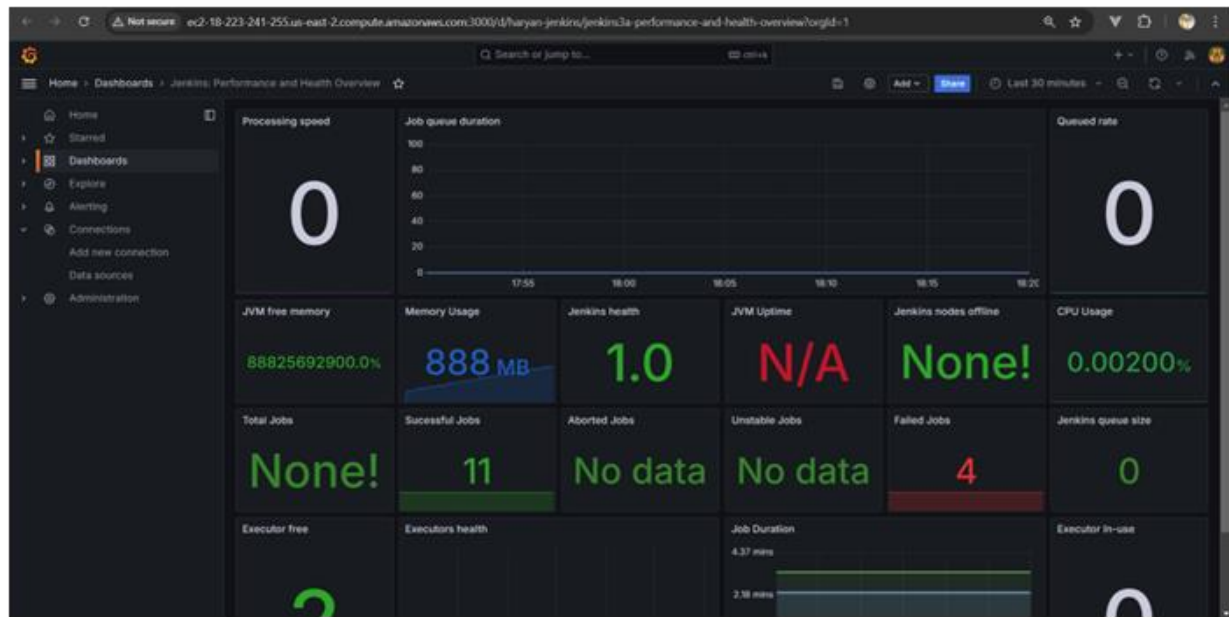


Figure 13: Build Analysis

Product Sample

Due to limited time and resources provided, along with a fully functional pipeline for developing and maintaining code, only a certain number of features have been fulfilled.

The example of the login page of the management system is demonstrated as below:

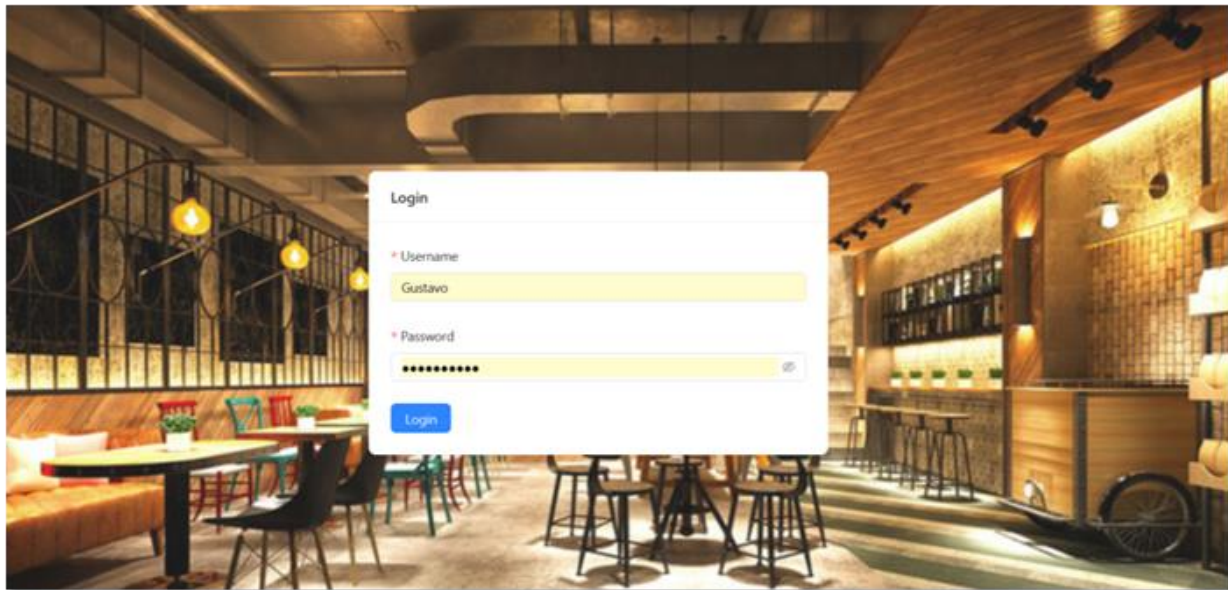


Figure 14: Login Page - Management System

After logged in, the user can choose which warehouse they want to manage, as follows:

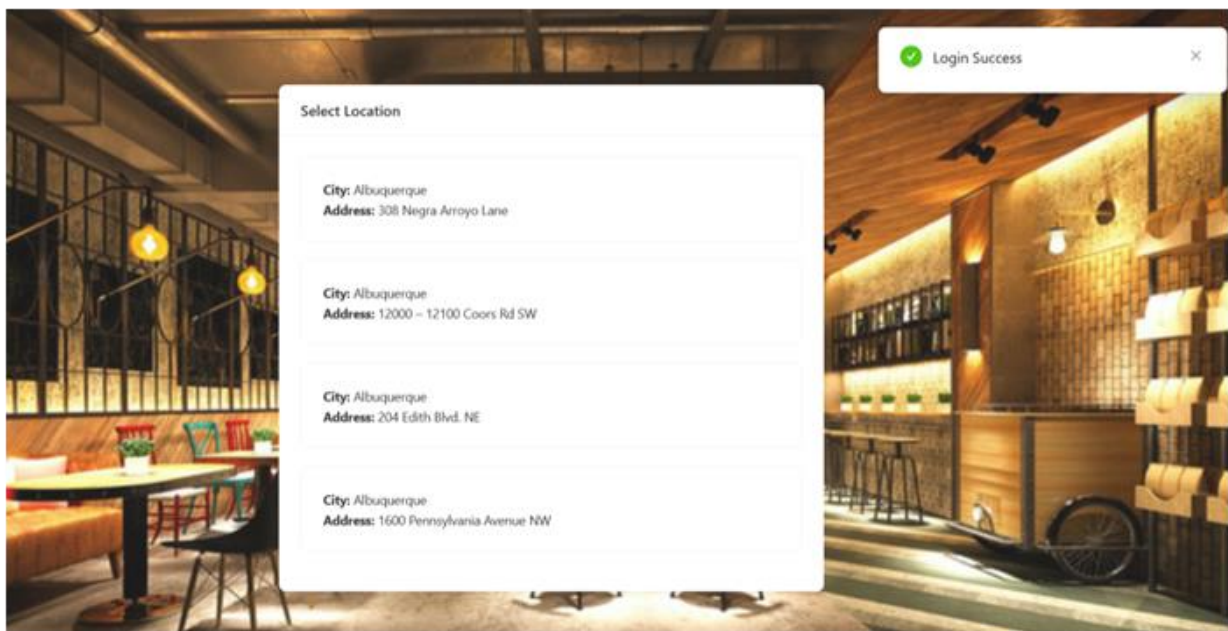
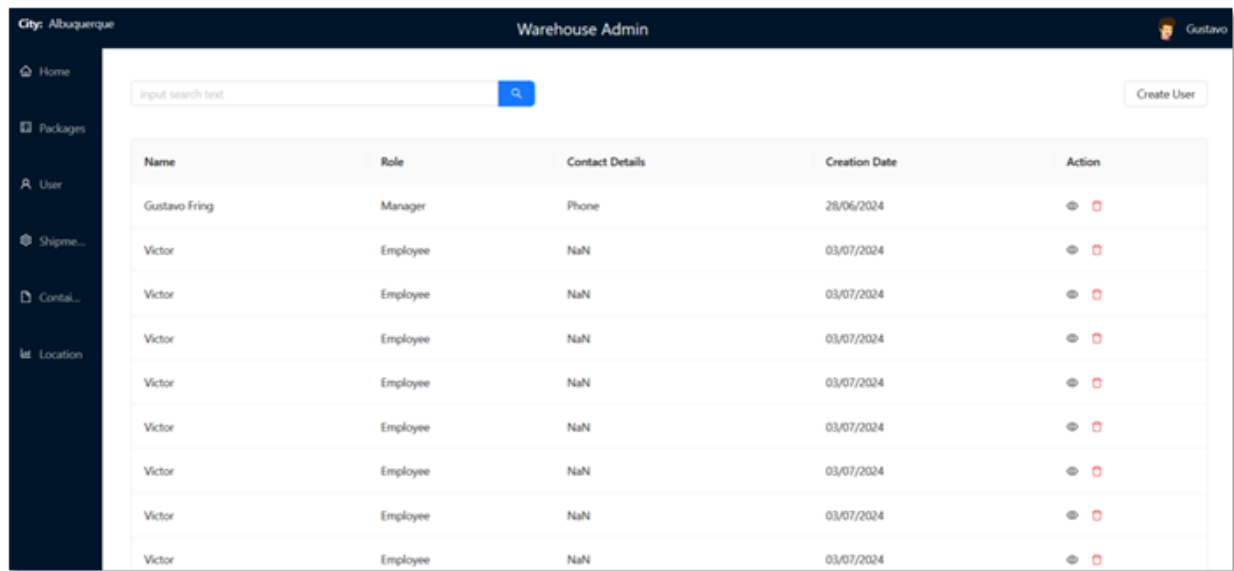


Figure 15: Select location

After choosing location, the manager can manage their employee, current package available, check shipment status, container status and adding other warehouse (location) as well.



The screenshot shows the 'Warehouse Admin' interface. On the left is a dark sidebar with navigation links: Home, Packages, User, Shipme..., Contal..., and Location. The main area has a top header with 'City: Albuquerque' and 'Warehouse Admin', and a user profile 'Gustavo'. Below the header is a search bar and a 'Create User' button. The central part of the page contains a table with the following data:



















Name	Role	Contact Details	Creation Date	Action
Gustavo Fring	Manager	Phone	28/06/2024	 
Victor	Employee	NaN	03/07/2024	 
Victor	Employee	NaN	03/07/2024	 
Victor	Employee	NaN	03/07/2024	 
Victor	Employee	NaN	03/07/2024	 
Victor	Employee	NaN	03/07/2024	 
Victor	Employee	NaN	03/07/2024	 
Victor	Employee	NaN	03/07/2024	 
Victor	Employee	NaN	03/07/2024	 

Figure 16: Warehouse Admin page

Project Outcomes

The successful implementation of our DevOps project has yielded several significant outcomes, enhancing the development and deployment processes for Gemadept Corporation's Smart Warehouse Management System (WMS). These outcomes are detailed below:

Improved Development Efficiency

- **Automated CI/CD Pipelines:** By leveraging Jenkins, GitHub Actions, and Docker, the project has established robust Continuous Integration and Continuous Deployment (CI/CD) pipelines. This automation has significantly reduced manual intervention, leading to faster and more reliable code integration and deployment cycles.
- **Consistent Environments:** Utilizing Docker for containerization ensures that the application runs consistently across different environments. This has minimized

discrepancies between development, staging, and production environments, reducing the "it works on my machine" issues.

- **Enhanced Code Quality and Security:** The integration of SonarQube, OWASP Scan, and Trivy into the CI/CD pipelines has provided continuous inspection of code quality and security. This has led to early detection of bugs, vulnerabilities, and security flaws, ensuring that only high-quality and secure code is deployed.

Optimized Resource Utilization

- **Scalable Infrastructure:** The use of Kubernetes for orchestrating containerized applications has enabled efficient scaling and management of resources. This ensures that the system can handle varying workloads without compromising performance or incurring unnecessary costs.
- **Efficient Resource Allocation:** By implementing an AWS infrastructure with services such as EC2, EKS, RDS, and S3, the project has optimized resource allocation. This has led to improved performance, reliability, and cost-effectiveness.

Enhanced Monitoring and Maintenance

- **Comprehensive Monitoring:** The integration of Prometheus and Grafana has provided detailed metrics on system performance and health. This has enabled real-time monitoring, proactive issue detection, and quick resolution, ensuring high system availability and reliability.
- **Preventive Maintenance:** Establishing a preventive maintenance schedule for equipment and infrastructure has reduced repair costs and downtime. This proactive approach ensures that potential issues are addressed before they escalate into major problems.

Streamlined Operations and Improved Collaboration

- **Real-Time Data Visibility:** IoT sensors, RFID technology, and advanced analytics integrated into the Smart WMS provide real-time visibility of inventory movement and location. This has streamlined operations, improved decision-making, and enhanced overall operational efficiency.
- **Improved Collaboration:** The use of version control with GitHub and the integration of CI/CD tools have facilitated better collaboration among development teams. This has

ensured that all team members are aligned and that code changes are tracked and integrated seamlessly.

Business Impact

- **Cost Reduction:** The optimized storage and retrieval processes, automated data capture, and preventive maintenance have collectively led to a significant reduction in operational costs. This has enhanced profitability and provided a competitive edge.
- **Increased Customer Satisfaction:** The improvements in order fulfillment processes, real-time tracking, and inventory accuracy have resulted in faster and more reliable order deliveries. This has increased customer satisfaction and strengthened customer relationships.
- **Future-Readiness:** The comprehensive approach to integrating advanced technologies and establishing robust DevOps practices has positioned Gemadept Corporation for future growth and success in the logistics sector.

Conclusion

In conclusion, the project has not only achieved the technical objectives but also delivered substantial business benefits. The deployment of the Smart Warehouse Management System, supported by a robust DevOps framework, has transformed Gemadept Corporation's warehouse operations, setting a new standard for efficiency, accuracy, and customer satisfaction.