INSTITUTE FOR ADVANCED

COMPUTING AND

SOFTWARE DEVELOPMENT

AKURDI, PUNE

DOCUMENTATION ON

"DevSecOps MERN Stack Application Deployment on AWS EKS Cluster"

PG-DITISS Aug-2024

SUBMITTED BY: -

GROUP NO: 02

AARTI LANKE (248402)

KANKSHA PATIL (248432)

MS. RUTUJA KULKARNI	**MR. ROHIT PURANIK**
PROJECT GUIDE	**CENTRE CO-ORDINATOR**

ACKNOWLEDGMENT

I would like to express my sincere gratitude to **Ms Rutuja Kulkarni** (Project Guide) and **Mrs. Sushma Hattarki** (Course Coordinator) my project guide, for her invaluable guidance, continuous support, and insightful feedback throughout the development of this **DevSecOps MERN Stack Application Deployment on AWS EKS Cluster** project. Her expertise and mentorship were instrumental in the successful completion of this work.

I am also deeply thankful to **Mr. Rohit Puranik**, our Centre Coordinator, for his support and encouragement, which provided the necessary direction and motivation to accomplish this project.

Furthermore, I would like to extend my appreciation to my colleagues and peers for their collaboration, constructive suggestions, and constant support during various stages of the project.

Their collective contributions have been crucial in the smooth execution and completion of this project.

Project Associates:

Aarti Lanke (248402)

Akanksha Patil (248432)

ABSTRACT

The project **"DevSecOps MERN Stack Application Deployment on AWS EKS Cluster"** focuses on automating the software development lifecycle while integrating security into every phase. It leverages DevSecOps practices to ensure secure, scalable, and resilient application deployment using Continuous Integration (CI) and Continuous Deployment (CD) methodologies. The MERN stack (MongoDB, Express.js, React.js, Node.js) is deployed on AWS Elastic Kubernetes Service (EKS) for cloud-native scalability and high availability. Terraform is used for Infrastructure as Code (IaC) to automate AWS resource provisioning, while Docker and Kubernetes manage containerized deployments. Jenkins orchestrates the CI/CD pipeline, integrating security tools such as SonarQube for static code analysis, OWASP Dependency-Check for vulnerability assessment, and Trivy for container security scanning. To strengthen security, Role-Based Access Control (RBAC), IAM policies, and network security measures are enforced. Secrets are securely managed using AWS Secrets Manager. Real-time monitoring with Prometheus and Grafana enhances observability and proactive threat detection. By embedding security into the Software Development Lifecycle (SDLC), this project demonstrates a robust, automated, and security-hardened deployment approach, aligning with industry best practices for modern cloud applications.

TABLE OF CONTENTS

1. INTRODUCTION

In modern software development, continuous integration and continuous deployment (CI/CD) have become essential practices to streamline the software release process. Traditional deployment methods often involve manual intervention, which is time-consuming, error-prone, and inefficient. Organizations now demand faster software delivery cycles with high reliability, security, and performance, making automation a necessity. This project focuses on implementing a CI/CD pipeline to automate the software development lifecycle, ensuring smooth integration, testing, and deployment while continuously monitoring application performance.

The project integrates a variety of DevOps tools, including Git for version control, Jenkins for pipeline automation, Maven for build management, Nexus for artifact repository, SonarQube for code quality analysis, and Trivy for security scanning. Docker is used for containerization, while Terraform automates infrastructure provisioning in AWS. Once deployed, Prometheus collects real-time metrics, and Grafana provides visualization and monitoring to detect performance issues and ensure system stability. By implementing this automated workflow, the project aims to enhance software delivery speed, improve security, and maintain operational efficiency.

2. PROBLEM STATEMENT

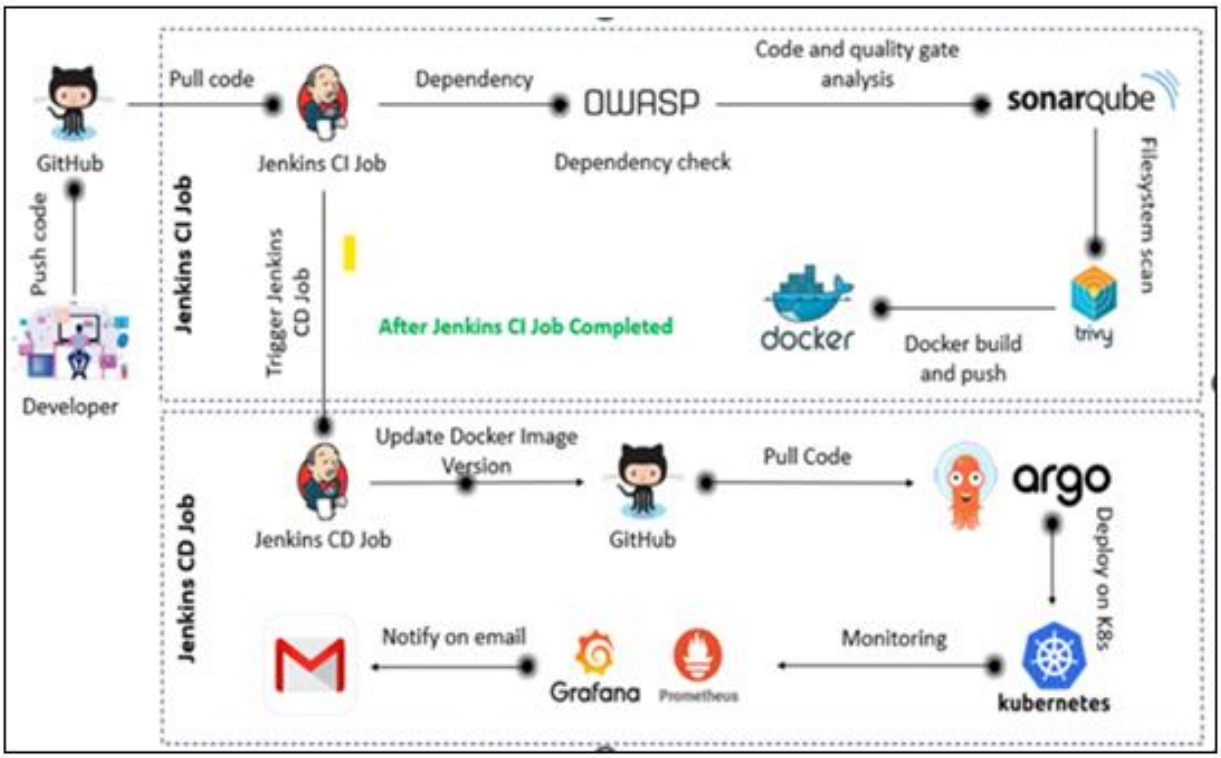
Traditional software deployment processes face significant challenges, including manual errors, security vulnerabilities, infrastructure inconsistencies, and lack of real-time monitoring. As applications grow in complexity and scale, ensuring security, automation, and scalability becomes increasingly difficult. Manual deployment methods are prone to human errors, lack of automated security checks, and make it harder to maintain stability while delivering rapid feature updates.

This project aims to address these challenges by implementing DevSecOps principles for deploying a MERN (MongoDB, Express.js, React.js, Node.js) stack application on AWS Elastic Kubernetes Service (EKS). The key challenges this project solves include:

1. **Security Integration in CI/CD** – Embedding security within the Software Development Lifecycle (SDLC) through automated vulnerability assessments and security compliance checks.
2. **Scalability and Performance** – Leveraging Kubernetes (EKS) for efficient workload management, auto-scaling, and high availability.
3. **Automated Security Compliance** – Integrating automated vulnerability scanning for source code, dependencies, and container images to detect and mitigate security risks.
4. **Access Control and Authentication** – Enforcing Role-Based Access Control (RBAC), IAM roles, and network security policies to prevent unauthorized access.
5. **Container Security** – Implementing best practices such as image scanning, least privilege policies, and runtime security monitoring for secure containerized deployment.
6. **Infrastructure as Code (IaC)** – Using Terraform or Helm for secure, automated provisioning and management of cloud infrastructure.
7. **Continuous Monitoring and Logging** – Setting up centralized logging, real-time monitoring, and alerting mechanisms to detect security incidents and performance issues proactively.

3.METHODOLOGY

3.1 Work-Flow: -



3.2 Detail Explanation: -

The methodology for deploying an application using a CI/CD pipeline and monitoring involves multiple phases, each integrating different DevOps tools to automate the software development lifecycle. The process follows a structured approach, ensuring seamless code integration, automated testing, secure artifact storage, containerized deployment, infrastructure provisioning, and continuous monitoring.

1. Source Code Management:

The development process begins with source code management using **Git**, where developers commit and push their changes to a centralized repository. A branching strategy is followed to ensure smooth collaboration and version control. GitHub or GitLab is used to store and manage the codebase.

2. Developer Pushes Code to GitHub:

The development process begins when a developer writes new code or modifies existing code and pushes the changes to a GitHub repository. This triggers an automated pipeline managed by Jenkins, initiating the Continuous Integration (CI) process. GitHub serves as the version control system, ensuring that the source code is managed efficiently.

3. Jenkins CI Job Execution (Continuous Integration Phase)

Once the code is pushed to GitHub, the **Jenkins CI job** automatically pulls the latest code and initiates the following steps:

a. Dependency Scanning with OWASP Dependency Check

- **OWASP Dependency-Check** is a security tool that scans project dependencies for known vulnerabilities.
- It identifies potential security risks associated with third-party libraries and dependencies used in the application.
- If vulnerabilities are detected, the pipeline can either fail or continue based on predefined security thresholds.

b. Static Code Analysis with SonarQube

- SonarQube is integrated into the CI pipeline to perform static code analysis.
- It checks for code quality, bugs, security vulnerabilities, and code smells before proceeding to the next stage.
- The pipeline enforces a quality gate, ensuring that the code meets the required standards before further processing.

c. Container Security Scanning with Trivy

- Trivy is a container security tool used to scan Docker images for vulnerabilities.
- It analyses the filesystem, OS packages, and application dependencies within the container.
- This step ensures that the built Docker images do not contain known security flaws that could be exploited in production.

4. Docker Build and Push to Registry

Once the security scans are successfully completed:

- Jenkins builds a Docker image from the application source code.
- The built image is tagged appropriately (e.g., with the latest commit ID or version number).
- The Docker image is pushed to a container registry (such as Docker Hub, AWS Elastic Container Registry (ECR), or another private registry).

This step ensures that the application is packaged into a consistent and portable containerized format that can be deployed across various environments.

5. Jenkins Triggers CD Job (Automated Deployment Process)

After the Jenkins CI job completes successfully, Jenkins automatically triggers the CD (Continuous Deployment) job to deploy the latest application version.

6. Jenkins CD Job Execution (Continuous Deployment Phase)

a. Updating the Docker Image Version in GitHub

- The newly built Docker image version is updated in the application's Kubernetes deployment configuration files (stored in GitHub).
- This ensures that the latest version is deployed whenever Kubernetes pulls the updated manifests.

b. GitOps-Based Deployment with ArgoCD

- ArgoCD follows the GitOps approach, where it continuously monitors the GitHub repository for any changes in the deployment configurations.
- When ArgoCD detects the updated Docker image version, it automatically pulls the changes and deploys the latest application version to the Kubernetes cluster on AWS EKS.
- This approach ensures that the desired state of the application is always maintained based on the configuration stored in GitHub.

7. Monitoring with Prometheus & Grafana

Once the application is deployed, real-time monitoring is essential to track performance and detect potential issues.

a. Prometheus – Metrics Collection

- Prometheus is used for collecting metrics from Kubernetes and application workloads.
- It gathers data related to CPU usage, memory consumption, response times, and system health.
- Prometheus is configured to raise alerts if certain thresholds are breached (e.g., high memory usage or application crashes).

b. Grafana – Visualization and Dashboards

- Grafana provides an intuitive dashboard for visualizing real-time metrics and logs collected by Prometheus.
- It helps DevOps teams analyze system performance, identify bottlenecks, and troubleshoot issues efficiently.
- This enables proactive system monitoring, reducing downtime and improving overall application reliability.

8. Notification and Reporting via Email

After deployment, the pipeline sends automated email notifications regarding the success or failure of the deployment.

- If the deployment is successful, the email contains details of the updated Docker image, deployment logs, and monitoring status.
- If the deployment fails, the email includes error logs, security scan reports, and troubleshooting steps.
- This ensures that the DevOps team is promptly informed and can take corrective action if needed.

4. REQUIREMENT SPECIFICATION

4.1 Hardware Requirements:

1. RAM: 8GB Minimum, 16 GB Recommended
2. HDD: 100GB free space for virtual machines

4.2 Software Requirements:

1. Operating System: ubuntu
2. Virtualization Tool: VMware Workstation Pro
3. Check Code Quality : SonarQube
4. Version Control: Git
5. Scanning Tools: Nessus, Trivy

4.3 Details about Software:

1. Version Control System – GitHub

- **Purpose:** Manages and stores source code while enabling collaboration among developers.
- **Tool Used:** GitHub.
- **Functionality:** Provides a **centralized repository** for code storage, version control, and history tracking. Supports **branching, merging, and pull requests** to facilitate teamwork. Triggers the **Jenkins CI pipeline** when new code is pushed.

2. CI/CD Automation – Jenkins

- **Purpose:** Automates the entire CI/CD pipeline, ensuring efficient code integration, testing, and deployment.
- **Tool Used:** Jenkins.
- **Functionality: Continuous Integration (CI):** Pulls code from GitHub, runs tests, performs security scans, and builds Docker images.**Continuous Deployment (CD):** Updates deployment files and triggers ArgoCD for automated Kubernetes deployment.**Pipeline Orchestration:** Manages the sequence of steps including security checks, builds, and monitoring integration.

3. Containerization & Orchestration – Docker & Kubernetes

a. Containerization – Docker

- **Purpose:** Packages applications into portable, lightweight containers for consistent deployment.
- **Tool Used:** Docker.
- **Functionality:** Creates isolated containers that run the application and its dependencies. Ensures the application runs the same way across different environments. Builds and pushes container images to a registry (Docker Hub or AWS ECR).

b. Orchestration – Kubernetes (K8s)

- **Purpose:** Manages containerized applications at scale, ensuring high availability and fault tolerance.
- **Tool Used:** Kubernetes.
- **Functionality: Automated Deployment & Scaling:** Ensures applications run efficiently with **auto-scaling and load balancing**. **Self-Healing:** Detects and restarts failed containers automatically. **Network & Security Policies:** Controls access and enforces security within containerized applications.

4. Dependency Security Scanning – OWASP Dependency-Check

- **Purpose:** Identifies known vulnerabilities in application dependencies.
- **Tool Used:** OWASP Dependency-Check.
- **Functionality:** Scans third-party libraries and dependencies for **CVE (Common Vulnerabilities and Exposures)**. Helps developers **patch vulnerable dependencies** before production deployment. Integrated into Jenkins to enforce security compliance during the CI phase.

5. Static Code Analysis Tool – SonarQube

- **Purpose:** Performs **code quality and security analysis** to ensure best practices are followed.
- **Tool Used:** SonarQube.
- **Functionality: Identifies bugs, code smells, and security vulnerabilities** in the source code. Ensures compliance with **coding standards and industry best practices**. Integrates with Jenkins to enforce **quality gates** before proceeding with deployment.

6. Container Vulnerability Scanning – Trivy

- **Purpose:** Scans Docker images and container filesystems for vulnerabilities.
- **Tool Used:** Trivy.
- **Functionality:** Detects security risks in **OS packages, dependencies, and containerized applications**. Performs both **static and runtime security scans**. Prevents deployment of vulnerable images in Kubernetes clusters.

7. Infrastructure as Code (IaC) – Terraform

- **Purpose:** Automates cloud infrastructure provisioning and management.
- **Tool Used:** Terraform.
- **Functionality:** Manages AWS resources such as **EKS, EC2, VPC, IAM roles, and security groups**. Enables **repeatable, consistent, and scalable** infrastructure setup. Provides **version control for infrastructure configurations**, making rollbacks easier.

8. Monitoring & Observability – Prometheus & Grafana

a. Monitoring – Prometheus

- **Purpose:** Collects real-time metrics and alerts based on system performance.

- **Tool Used:** Prometheus.
- **Functionality: Scrapes metrics** from Kubernetes clusters and application services. Detects anomalies and **triggers alerts** for CPU usage, memory consumption, or network failures. Stores historical data for **trend analysis**.

b. Visualization – Grafana

- **Purpose:** Provides real-time dashboards for system monitoring and analytics.
- **Tool Used:** Grafana.
- **Functionality:** Displays **visual dashboards** based on Prometheus metrics. Helps identify **performance bottlenecks and application failures**. Provides alerts and notifications for quick issue resolution.

9. MERN Stack (MongoDB, Express.js, React.js, Node.js)

a. MongoDB (Database)

- **Purpose:** NoSQL database to store and manage application data.
- **Functionality:** Handles structured and unstructured data efficiently. Supports high availability and scalability with **replication and sharding**. Manages user authentication, application sessions, and API data storage.

b. Express.js (Backend Framework)

- **Purpose:** Lightweight and fast Node.js framework for building APIs.
- **Functionality:** Manages server-side routing and middleware execution. Handles RESTful API requests and responses. Ensures backend scalability and security.

c. React.js (Frontend Framework)

- **Purpose:** Provides an interactive and responsive UI for the application.
- **Functionality:** Creates reusable components for better code management. Ensures high performance through **Virtual DOM rendering**. Fetches data from backend APIs dynamically.

d. Node.js (Runtime Environment)

- **Purpose:** Executes JavaScript on the backend for seamless full-stack development.
- **Functionality:** Manages concurrent user requests efficiently. Handles API requests and database interactions. Supports event-driven and non-blocking architecture for high-speed performance

5. IMPLEMENTATION

Implementation of DevSecOps Mega Project (MERN Stack Deployment on AWS EKS)

This project follows a **DevSecOps approach** to deploy a **MERN (MongoDB, Express.js, React.js, Node.js) stack application** on **AWS Elastic Kubernetes Service (EKS)** using automated CI/CD pipelines, security integration, and monitoring tools. Below is the detailed breakdown of the implementation.

1. Application Layer (MERN Stack) & Version Control (GitHub & Shared Library)

- **Technologies Used:**
  - **MongoDB** – Database for storing application data.
  - **Express.js** – Backend framework for handling API requests.
  - **React.js** – Frontend framework for building a user-friendly interface.
  - **Node.js** – Runtime environment to execute JavaScript on the server side.
  - **GitHub** – Version control system for managing source code.
- **Implementation Steps:**
  - Developers push the MERN stack application source code to **GitHub**.
  - A **GitHub Shared Library** is utilized to store reusable CI/CD pipeline scripts, reducing redundancy and improving maintainability across multiple projects.
  - The CI/CD pipeline is triggered via Jenkins, pulling the latest code.
  - Application code is **containerized using Docker** for portability.

2. Master Machine Setup (Infrastructure as Code using Terraform)

- **Tools Used:**
  - **Terraform** – Automates cloud resource provisioning.
  - **AWS IAM (Identity & Access Management)** – Manages security roles and permissions.
  - **Key Pair & Security Groups** – Provides SSH access and firewall rules.
  - **Master Node** – Manages Kubernetes cluster and deployment automation.
- **Implementation Steps:**

- Terraform scripts are written to provision AWS infrastructure (EKS, IAM, Security Groups, Key Pair).
- IAM roles and permissions are configured for **Jenkins, ArgoCD, and Kubernetes services**.
- Security groups ensure secure networking between application components.

3. Kubernetes Cluster on AWS EKS

- **Components Used:**
  - **AWS EKS (Elastic Kubernetes Service)** –Managed Kubernetes for container orchestration.
  - **Node 1 & Node 2** – Worker nodes running application containers.
- **Implementation Steps:**
  - Terraform provisions an **EKS cluster with auto-scaling worker nodes**.
  - Kubernetes resources (Deployments, Services, ConfigMaps, Secrets) are defined in YAML files.
  - The MERN application is deployed as Kubernetes **Pods and Services**.

4. DevSecOps Integration (CI/CD Security & Automation)

- **Tools Used:**
  - **Jenkins** – Automates Continuous Integration (CI) & Continuous Deployment (CD).
  - **Trivy** – Scans Docker images for vulnerabilities.
  - **OWASP Dependency-Check** – Scans third-party libraries for security vulnerabilities.
  - **SonarQube** – Performs static code analysis for bugs and vulnerabilities.
  - **Docker** – Containerizes the application.
  - **Mail Notification** – Sends deployment success/failure alerts.
- **Implementation Steps:**
  - **Jenkins CI/CD Pipeline Execution:**
    - Pulls code from GitHub.
    - Runs OWASP Dependency-Check to identify insecure dependencies.

        - Uses SonarQube for code quality and security analysis.
        - Trivy scans Docker images before deployment.
        - Builds the Docker image and pushes it to a container registry (Docker Hub, AWS ECR, etc.).
    - Jenkins triggers the CD process:
        - Updates the Kubernetes manifest file with the new Docker image version.
        - Pushes updated configurations to the Git repository.

5. GitOps for Continuous Deployment

- **Tools Used:**
    - **GitHub (Git Repository)** – Stores Kubernetes deployment manifests.
    - **ArgoCD** – Deploys application changes automatically.
- **Implementation Steps:**
    - ArgoCD monitors the Git repository for changes in Kubernetes manifests.
    - Pulls the latest deployment configuration and applies changes to the EKS cluster.
    - Ensures that the deployed application version matches the desired state in Git.

6. Monitoring & Observability

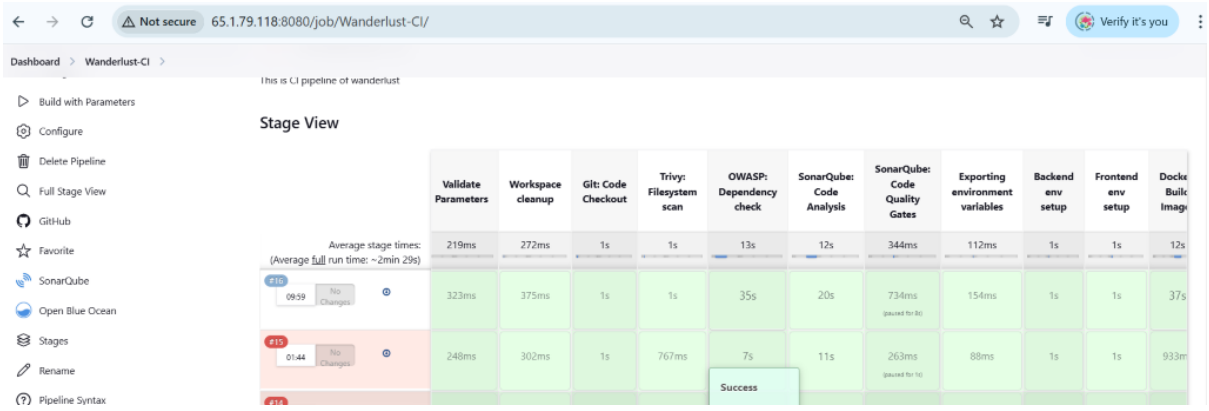
- **Tools Used:**
    - **Prometheus** – Collects system metrics and performance data.
    - **Grafana** – Provides real-time visualization and alerts.
- **Implementation Steps:**
    - Prometheus is installed in the Kubernetes cluster to monitor CPU, memory, request latency, and container health.
    - Grafana dashboards visualize real-time metrics for troubleshooting and optimization.
    - Alerts notify the team in case of failures or high resource usage.

Final Workflow Summary

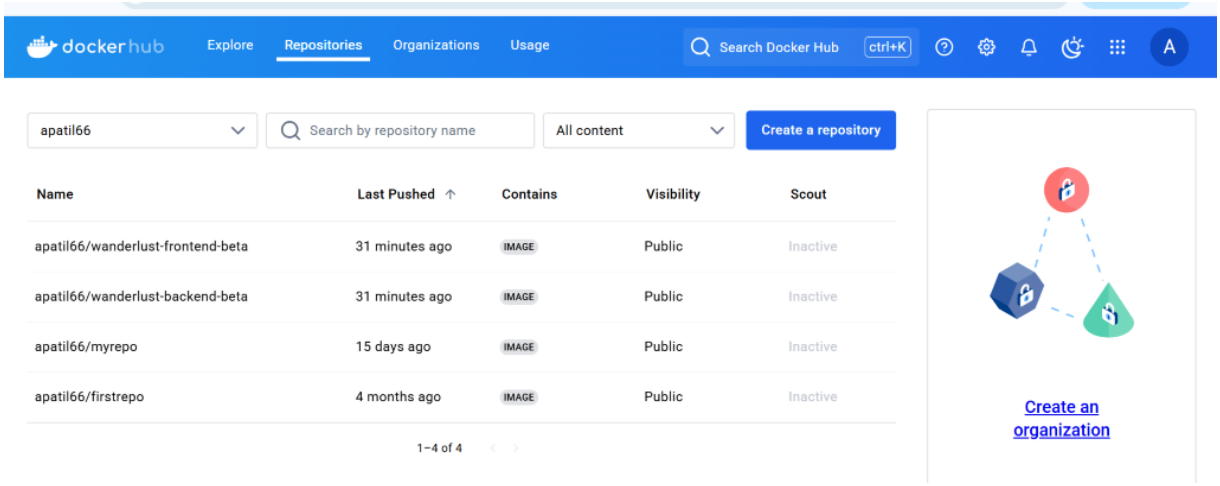
1. **Developers push code to GitHub** → Triggers **Jenkins CI/CD pipeline**.
2. **CI Pipeline:**
    - OWASP dependency check.
    - SonarQube static code analysis.
    - Trivy container vulnerability scanning.
    - Docker image build & push to registry.
3. **CD Pipeline:**
    - Updates deployment manifests in GitHub.
    - ArgoCD deploys the updated application to Kubernetes.
4. **Monitoring & Notifications:**
    - Prometheus & Grafana monitor system performance.
    - Jenkins sends email notifications on deployment success or failure.

6. APPLICATION

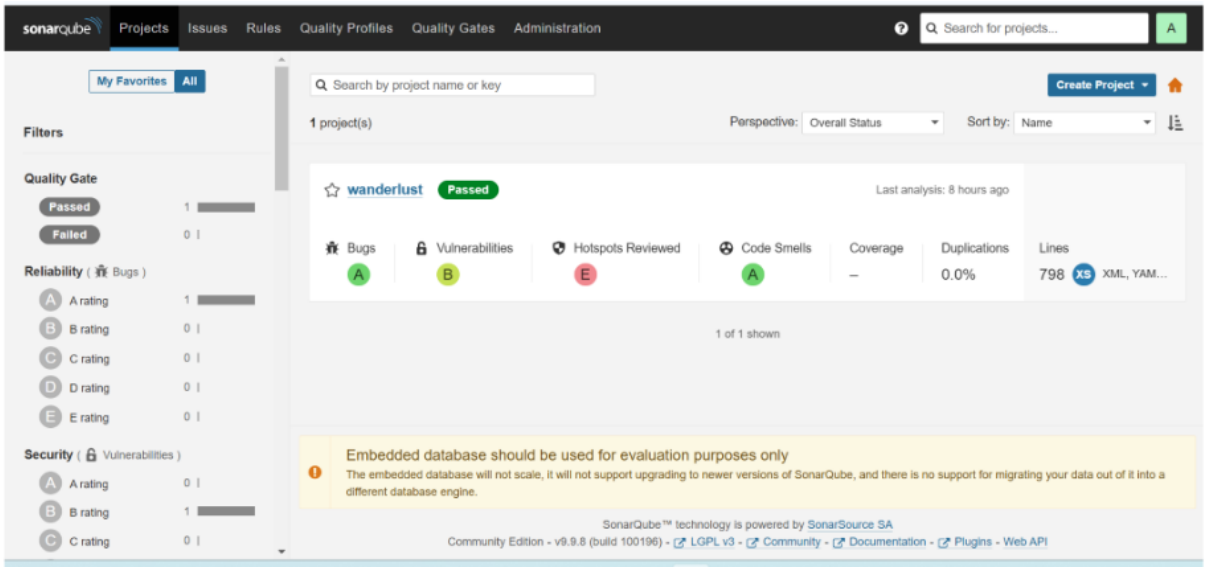
1. Jenkins pipeline



2. Docker repository



3. SonarQube setup



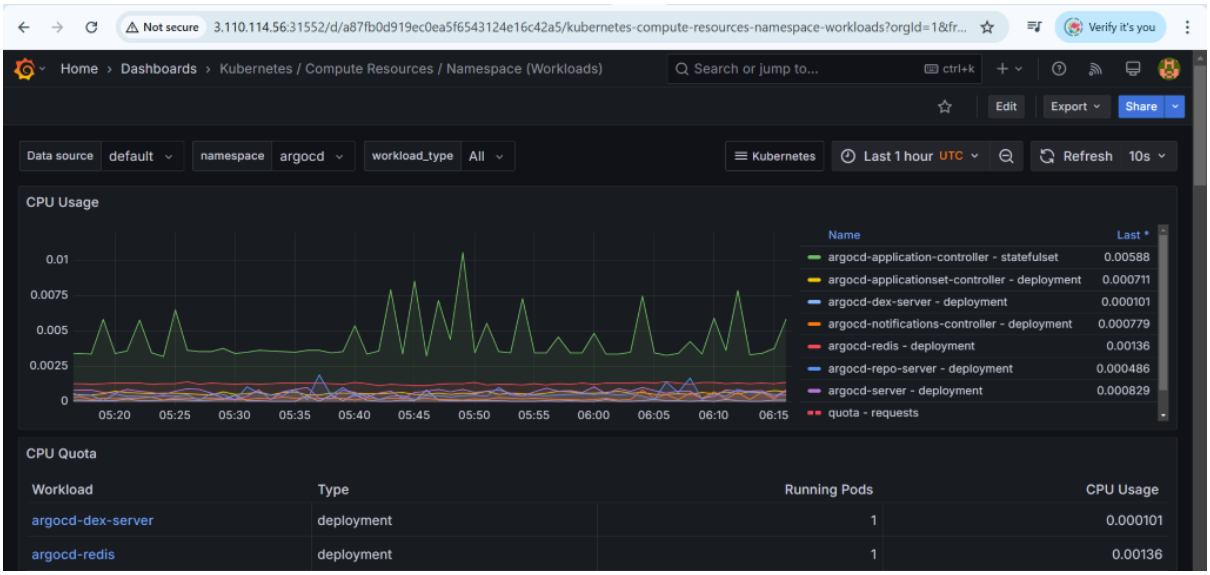
4. Trivy setup

MERN-stack-app-deployment/frontend/package-lock.json (npm)

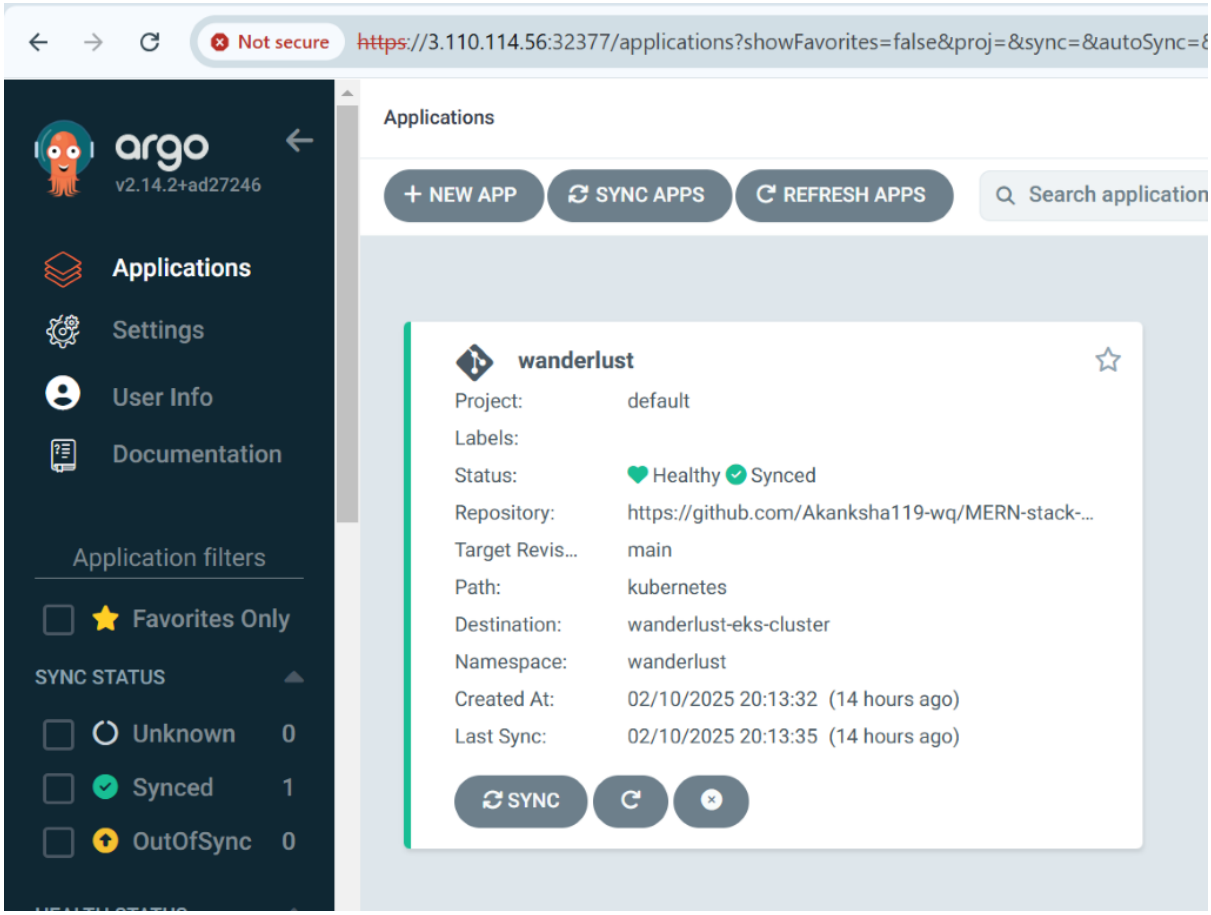
Total: 6 (UNKNOWN: 0, LOW: 0, MEDIUM: 4, HIGH: 2, CRITICAL: 0)

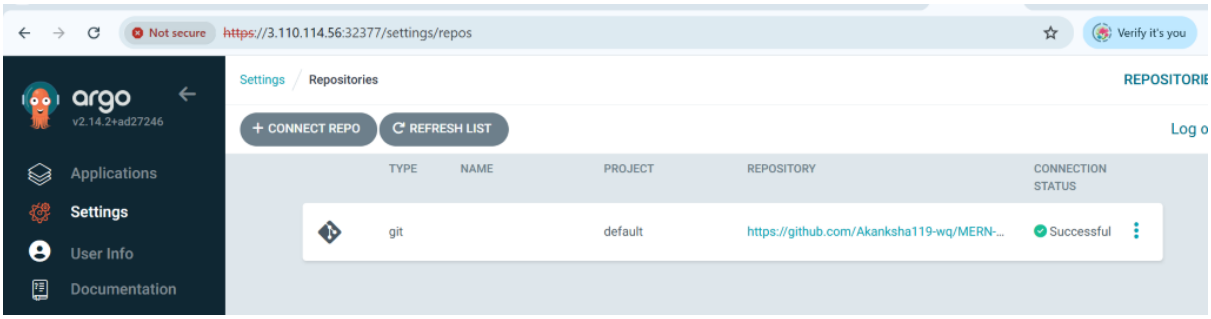
Library	Vulnerability	Severity	Status	Installed Version	Fixed Version	Title
axios	CVE-2024-39338	HIGH	fixed	1.6.1	1.7.4	axios: axios: Server-Side Request Forgery https://avd.aquasec.com/nvd/cve-2024-39338
braces	CVE-2024-4068			3.0.2	3.0.3	braces: fails to limit the number of characters it can handle https://avd.aquasec.com/nvd/cve-2024-4068
follow-redirects	CVE-2023-26159	MEDIUM		1.15.3	1.15.4	follow-redirects: Improper Input Validation due to the improper handling of URLs by... https://avd.aquasec.com/nvd/cve-2023-26159
	CVE-2024-28849				1.15.6	follow-redirects: Possible credential leak https://avd.aquasec.com/nvd/cve-2024-28849
micromatch	CVE-2024-4067			4.0.5	4.0.8	micromatch: vulnerable to Regular Expression Denial of Service https://avd.aquasec.com/nvd/cve-2024-4067

5. Grafana

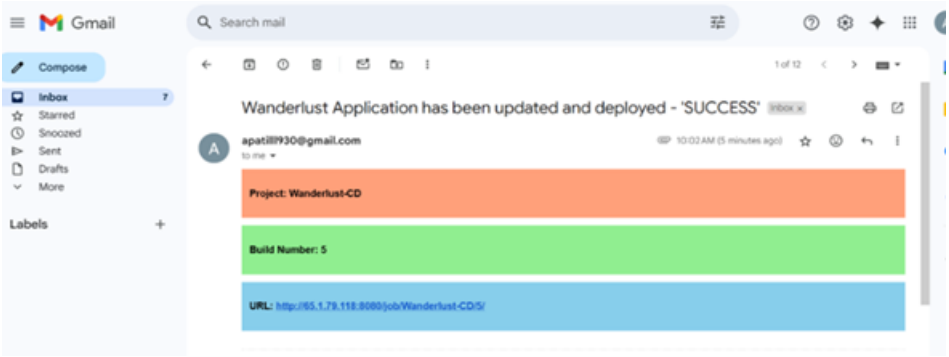


6. ArgoCD setup

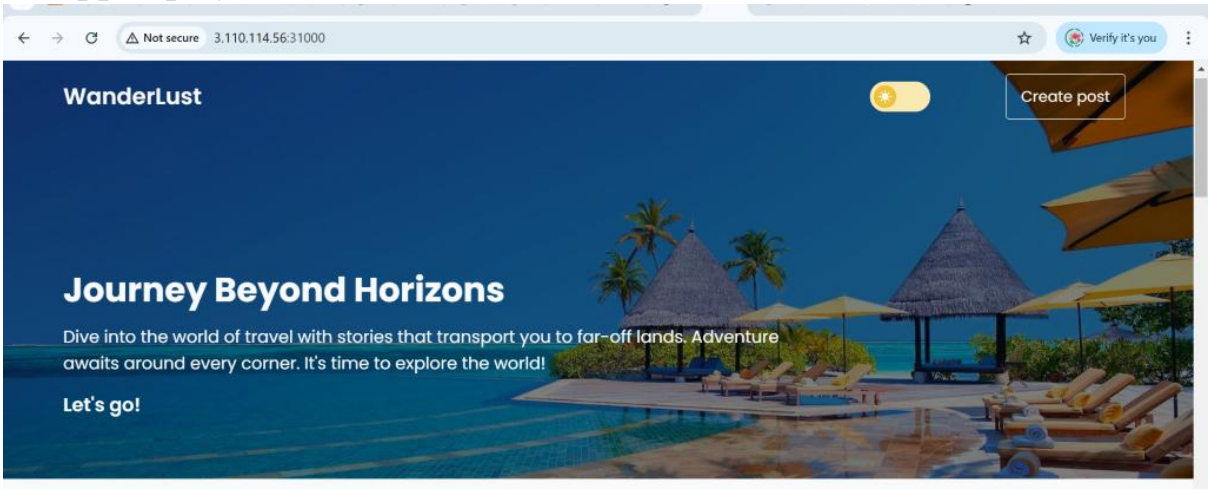




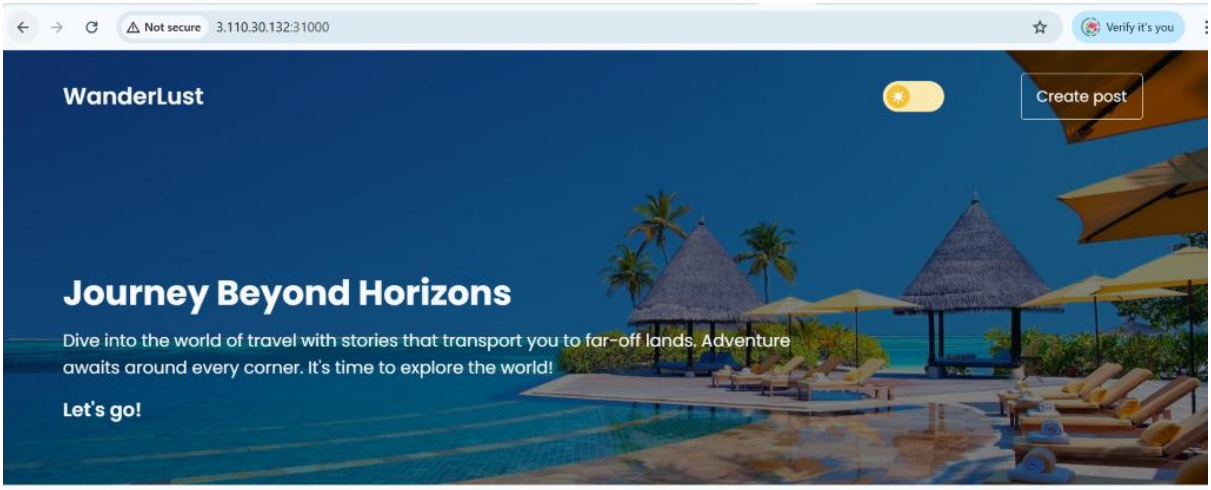
7. Mail notification after build successful



8. App deployment on node 1



9. App deployment on node 2



7. VULNERABILITY SCANNING

1. Static Code Analysis with SonarQube

- **Purpose:** Detects security vulnerabilities, bugs, and code quality issues in the source code.
- **Implementation:**
  - **SonarQube** is integrated into the **Jenkins pipeline** to scan the code after it is fetched from the Git repository. o It analyses the source code against predefined security rules and coding standards.
  - Generates a report highlighting security issues, including:
    - **SQL injection risks**
    - **Cross-Site Scripting (XSS) vulnerabilities**
    - **Hardcoded secrets (API keys, passwords, etc.)**
  - If the scan detects **critical vulnerabilities**, the pipeline halts further execution, preventing insecure code from moving to the next stage.

2. Container Security Scanning with Trivy

- **Purpose:** Scans Docker images for known security vulnerabilities before deployment.
- **Implementation:**
  - **Trivy** is configured in the Jenkins pipeline to automatically scan the Docker images after they are built. o It checks for **Common Vulnerabilities and Exposures (CVEs)** in:
    - **OS packages (Ubuntu, Alpine, etc.)**
    - **Application dependencies**
  - Generates a detailed report categorizing vulnerabilities as **Critical, High, Medium, or Low severity**.
  - If critical vulnerabilities are found, the pipeline stops deployment until they are patched. o Developers are notified, allowing them to update insecure dependencies and rebuild the container.

3. Automating Security Checks in CI/CD Pipeline

- **Pipeline Configuration:**
    - The **Jenkins pipeline** includes **SonarQube** and **Trivy** as mandatory security checkpoints.
    - If any tool detects **high or critical vulnerabilities**, the deployment is **blocked**, ensuring that only secure builds are deployed. o
    - The security reports generated by Trivy and SonarQube are archived for auditing and compliance tracking.

4. Continuous Monitoring for Security Issues

- **Purpose:** Detects vulnerabilities in running containers and cloud infrastructure.
- **Implementation:**
    - **Prometheus and Grafana** are configured to monitor container behaviour and system performance.
    - Alerts are triggered if unusual activity or **suspicious network traffic** is detected.
    - AWS security services (e.g., AWS Guard Duty, AWS Security Hub) can be integrated for additional threat detection.

8. ADVANTAGES & LIMITATIONS

8.1 Advantages: -

- **Improved Code Quality & Security**
  - **SonarQube** analyses the code for vulnerabilities, ensuring high-quality and secure code. o **Trivy** scans container images, detecting known vulnerabilities before deployment.
- **Efficient Infrastructure Management with Terraform**
  - **Terraform** enables automated, scalable, and reproducible cloud infrastructure provisioning.
  - Reduces manual errors in infrastructure setup and deployment.
- **Enhanced Monitoring & Observability**
  - **Prometheus** collects system metrics, while **Grafana** provides real-time visualization and alerting.
  - Helps in proactive issue detection and quick resolution of system failures.
- **Scalability & Flexibility**
  - The solution is highly scalable as it runs on **AWS cloud infrastructure** with support for auto-scaling.
  - **Docker containers** ensure that applications can run consistently in different environments.
- **Security Compliance & Auditability**
  - The pipeline enforces security scanning and maintains logs for compliance audits.
  - AWS security features and IAM policies ensure proper access control.

8.2 Limitations: -

- **High Resource Utilization**
  - Running Jenkins, SonarQube, Prometheus, and other services consumes system resources. ○ Additional AWS costs are incurred for cloud infrastructure and monitoring tools.
- **Potential Pipeline Failures**
  - If a single step in the CI/CD pipeline fails (e.g., code quality check, security scan), deployment is blocked.
  - Developers must frequently troubleshoot and resolve pipeline issues.
- **Security Risks in Misconfiguration**
  - Improper Terraform or IAM role configurations in AWS can lead to security vulnerabilities. ○ Unauthorized access to Nexus or Docker images can result in supply chain attacks.

9.CONCLUSION

The implementation of this **DevSecOps Mega Project** successfully automates the deployment of a **MERN stack application** on **AWS Elastic Kubernetes Service (EKS)** while integrating security, scalability, and monitoring. By leveraging modern DevSecOps tools and practices, this project ensures a secure and efficient software delivery lifecycle.

Key takeaways from this implementation include:

1. **Security Integration** – The project embeds security at every stage of the CI/CD pipeline using **OWASP Dependency-Check, SonarQube, and Trivy**, reducing the risk of vulnerabilities in the application and infrastructure.
2. **Automation & Scalability** – Infrastructure provisioning and application deployment are automated using **Terraform, Jenkins, and ArgoCD**, ensuring consistency, scalability, and efficiency in the deployment process.
3. **Continuous Monitoring** – The integration of **Prometheus and Grafana** provides real-time observability, helping to detect and mitigate performance and security issues proactively.
4. **GitOps-Driven Deployments** – Using **ArgoCD** for GitOps ensures that application deployments remain version-controlled, reproducible, and rollback-friendly.
5. **Efficient CI/CD Workflow** – Jenkins automates **build, security checks, testing, and deployment**, reducing manual effort and ensuring faster, error-free software releases.

By combining **DevOps, Security (DevSecOps), and GitOps** principles, this project provides a **robust, scalable, and secure** software delivery pipeline. It significantly reduces manual intervention, enhances security posture, and ensures that the application remains resilient and high-performing in cloud-native environments.

This approach aligns with industry best practices, making it an ideal model for modern cloud-based application deployments.

10.REFERENCES

1. https://github.com/Akanksha119-wq/MERN-stack-app-deployment
2. https://hub.docker.com/u/apatil66
3. https://docs.sonarsource.com/sonarqube-server/10.8/