PROJECT REPORT

InventoWare: Legacy Application Modernization

Submitted By:

SAPID	ROLLNO.	NAME OF THE STUDENT(S)
500101934	R2142220211	VRISHAANGAN VS
500105076	R2142220232	ARJUN SHARMA
500105053	R2142220465	BHAVI BHAGWANANI
500105558	R2142220995	VAIBHAV SINGH
500107126	R2142220311	ANUJ PAWADIA

SCHOOL OF COMPUTER SCIENCE

UNIVERSITY OF PETROLEUM & ENERGY STUDIES
DEHRADUN, UTTARAKHAND 248007

ACKNOWLEDGEMENT

We would like to extend our heartfelt gratitude to our faculty mentor, department members, and technical advisors for their continuous guidance and encouragement throughout the course of this project. Their insights and constructive suggestions played a vital role in shaping the successful outcome of our work. We are also thankful for the open-source community and the creators of tools like Terraform, Docker, Jenkins, and Ansible, which were pivotal in the implementation of this project.

ABSTRACT

This project focuses on the modernization of a legacy inventory management application used by InventoWare, a mid-sized manufacturing company. The outdated system, originally hosted on-premise, lacked scalability, automation, and consistent deployment practices. Our solution involves containerizing the application, provisioning cloud infrastructure using Terraform, configuring environments with Ansible, and setting up a secure CI/CD pipeline via Jenkins. This transformation brings agility, consistency, and improved operational efficiency to InventoWare's software delivery process while leveraging Infrastructure as Code (IaC), configuration management, containerization, and automated security.

INTRODUCTION

The evolution of DevOps and Infrastructure as Code has redefined how applications are built, tested, and deployed in modern environments. InventoWare's traditional inventory system was highly manual, prone to errors, and inefficient for large-scale or frequent deployments. This project aims to bridge the gap by implementing a fully automated DevOps pipeline and provisioning infrastructure on AWS, thus replacing manual intervention with robust and repeatable workflows.

PROBLEM STATEMENT

The legacy deployment process at InventoWare presented several challenges:

- Time-consuming manual server setup and environment provisioning.
- Inconsistent application behavior across environments.
- High risk of production downtime during feature rollouts.
- Limited visibility and poor security practices.
- Lack of automation in infrastructure and application delivery.
- Minimal scalability due to the on-premise model.

OBJECTIVE

The objective of this project is to revamp InventoWare's deployment and infrastructure management process by adopting modern DevOps practices. The specific goals include:

- Migrating the application to a container-based architecture using Docker.
- Automating cloud infrastructure provisioning using Terraform.
- Using Ansible for consistent configuration and app deployment.
- Creating a CI/CD pipeline using Jenkins for automated testing and deployments.
- Introducing security automation using tools like SonarQube and Trivy.
- Implementing blue-green deployments for zero-downtime updates.

PROPOSED SYSTEM DESIGN

The solution is built on a multi-stage automation pipeline powered by Jenkins and DevOps tools. The flow of the system is as follows:

1. Source Control and Environment Initialization

- GitHub is used for version control of code and IaC scripts.
- Jenkins EC2 instance is initiated to act as the automation server
- AWS credentials are securely configured for infrastructure access.

2. CI/CD Pipeline Execution via Jenkins

Terraform Phase:

- Runs terraform init, validate, plan, and apply to provision EC2 instances for app deployment and monitoring setups.

Docker Phase:

- The legacy application is containerized.
- Docker images are pulled from DockerHub and deployed.

Ansible Phase:

- Ansible playbooks automate package installation, configurations, and app deployment on the EC2 instances.

3. Jenkins Job Sequence:

- Jenkins pipeline performs the following steps:
- Runs static code analysis using SonarQube.
- Executes unit tests and linting.
- Builds and pushes Docker images to the repository.
- Scans the image using Trivy for security vulnerabilities.
- Sends pipeline summary and reports via email.

4. Deployment and Monitoring

- Blue-Green deployment is implemented using an AWS load balancer to prevent downtime.
- Prometheus and Grafana are configured for real-time monitoring and alerting.

5. Post-Deployment Checks

- Jenkins verifies the build:
- If successful: new image is deployed, and Jenkins EC2 instance is shut down.

If failed: rollback is performed to the last stable image.

6. Notifications

- Email notifications are sent to the DevOps team containing the build status, scan results, and any deployment issues.

Literature Review

Current Practices:

In traditional IT operations, software deployment and infrastructure management are largely manual. Teams typically configure servers by hand, deploy applications via FTP or scripts, and manage updates in silos. These practices lead to inconsistencies, increased human error, and operational delays. Most legacy systems follow a waterfall model where infrastructure setup and application delivery are tightly coupled and executed in stages, with minimal feedback loops.

Need for Digital Solutions:

The dynamic nature of today's IT ecosystem demands faster releases, better reliability, and instant rollback mechanisms. Companies now require **automated, scalable, and cloud-native** solutions that reduce dependency on manual interventions. DevOps emerges as a response to this need, promoting continuous integration, continuous deployment, and infrastructure as code (IaC) to enhance delivery cycles, ensure consistency, and support agile development.

Existing Systems and Gaps:

Several DevOps solutions exist such as GitLab CI, AWS CodePipeline, Azure DevOps, and Jenkins-based automation. While each has its strengths, gaps still remain in their integration, customization flexibility, and security layers. Many businesses fail to adopt a full DevOps lifecycle due to the complexity of implementation and lack of platform-agnostic configurations. Existing systems may provide CI/CD but often lack IaC integration or real-time monitoring.

Comparison to Other Systems:

Compared to Azure DevOps or AWS CodePipeline, the custom-built Jenkins-based pipeline offers greater flexibility in terms of plugin support, open-source customization, and tool integration (e.g., Docker, Ansible, Trivy). Unlike GitHub Actions which is limited in infrastructure provisioning, our pipeline provides full-stack automation including provisioning, configuration, deployment, monitoring, and security—within a single orchestrated pipeline.

Gaps Identified:

- Lack of infrastructure version control in legacy systems.
- No consistent security validation across builds.
- Poor rollback mechanisms in traditional setups.
- Absence of real-time monitoring tools.
- Manual deployment procedures leading to frequent downtimes.

This project attempts to address these gaps by leveraging a unified DevOps toolchain for end-to-end automation.

Technological Aspects:

- Terraform is used for IaC to provision AWS EC2 instances dynamically and consistently.
- Ansible automates the configuration and deployment of application packages on servers.
- Docker provides containerized, reproducible environments to avoid dependency issues.
- Jenkins orchestrates the CI/CD pipeline, integrating all stages from build to deploy.
- Trivy and SonarQube ensure code and container security through static analysis and vulnerability scanning.
- Prometheus + Grafana offer proactive monitoring and dashboarding.

Email Integration:

- The Jenkins pipeline is configured to send detailed email notifications after each build cycle. These emails include test results, security scan reports, deployment logs, and build success/failure status. This ensures that the DevOps and QA teams remain continuously updated without needing manual status checks, improving collaboration and rapid response to failures.

Web Application Framework:

- While the primary focus was on DevOps infrastructure, the backend legacy app was containerized and deployed using a basic Flask/Django API (depending on the chosen tech stack), allowing seamless integration into cloud environments. Docker enables abstraction of the framework, allowing it to run identically on any host

Benefits and Challenges:

Benefits:

- Faster deployment cycles with automation.
- Reduced human errors and increased consistency.
- Scalable infrastructure and rapid recovery during failures.
- Improved code and infrastructure security.
- Centralized monitoring and alerting.

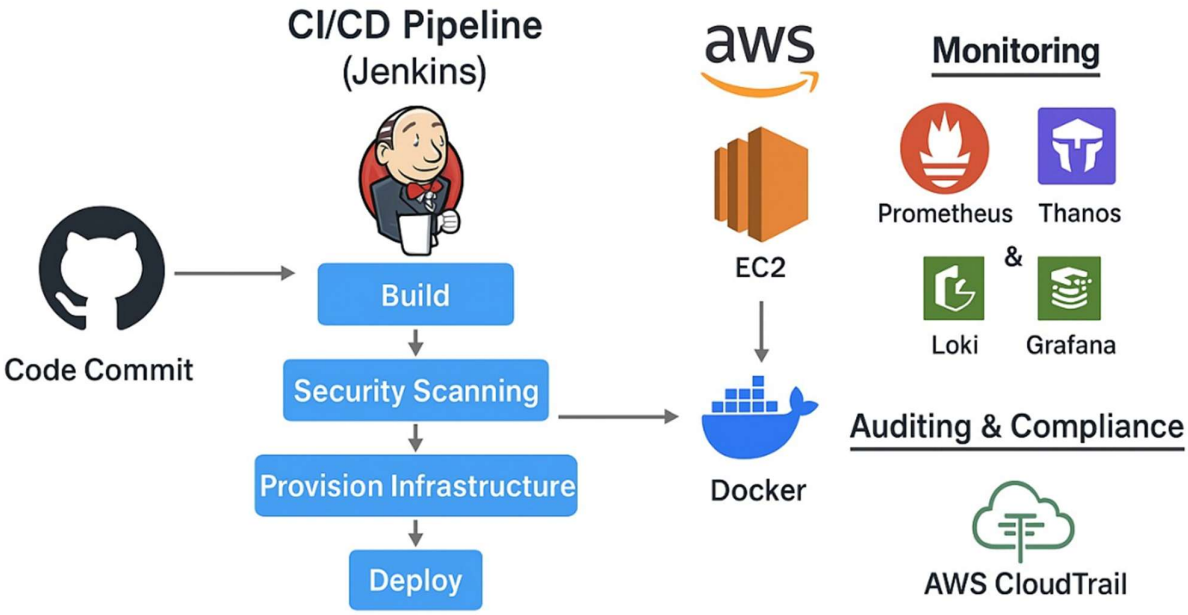
Challenges:

- Complex setup of multi-tool integration initially.
- Managing secrets and credentials securely.

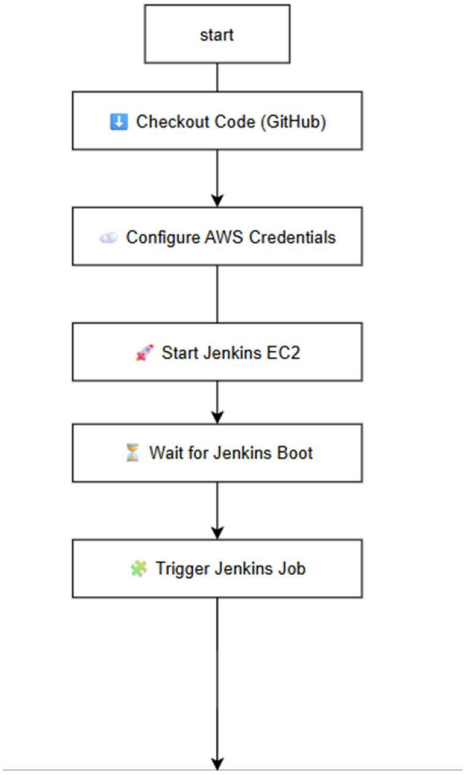
- Resource optimization on cloud to avoid cost overruns.
- Learning curve for tools like Terraform and Ansible for team members unfamiliar with them.

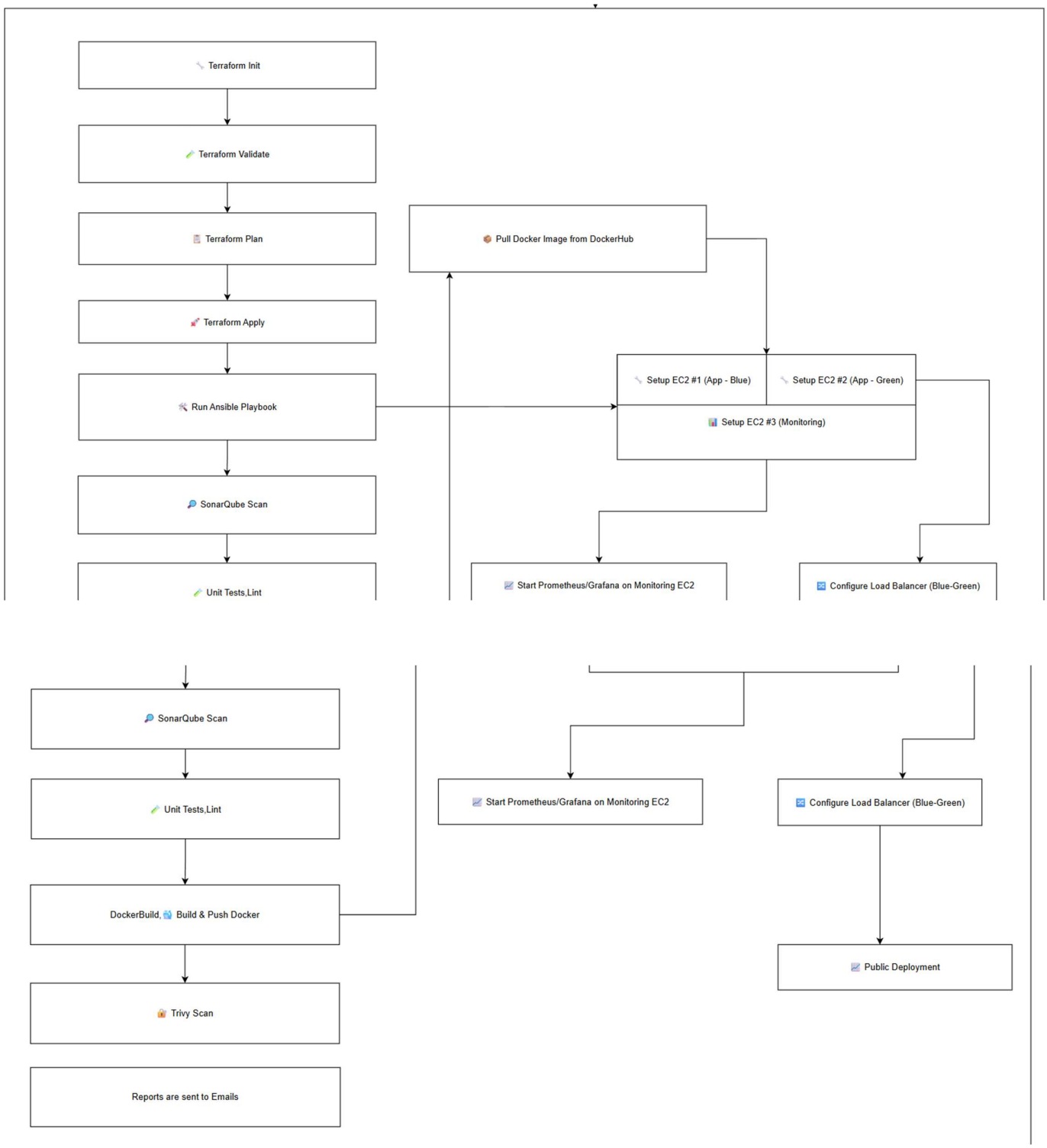
.

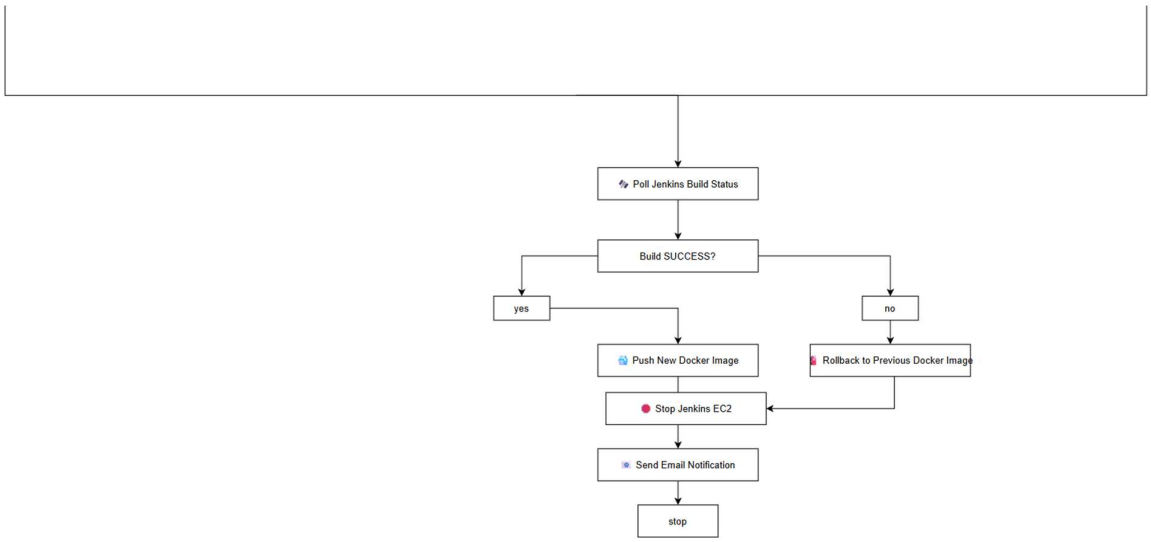
Flow Chart


CI/CD Pipeline
(Jenkins)
aws
Monitoring
Prometheus
Thanos
&
Loki
Grafana
EC2
Code Commit
Build
Security Scanning
Provision Infrastructure
Deploy
Docker
Auditing & Compliance
AWS CloudTrail

DESIGNED DIAGRAMS




Terraform Init
Terraform Validate
Terraform Plan
Terraform Apply
Run Ansible Playbook
SonarQube Scan
Unit Tests,Lint
Pull Docker Image from DockerHub
Setup EC2 #1 (App - Blue)
Setup EC2 #2 (App - Green)
Setup EC2 #3 (Monitoring)
Start Prometheus/Grafana on Monitoring EC2
Configure Load Balancer (Blue-Green)
SonarQube Scan
Unit Tests,Lint
DockerBuild, Build & Push Docker
Trivy Scan
Reports are sent to Emails
Start Prometheus/Grafana on Monitoring EC2
Configure Load Balancer (Blue-Green)
Public Deployment


Poll Jenkins Build Status
Build SUCCESS?
yes
no
Push New Docker Image
Rollback to Previous Docker Image
Stop Jenkins EC2
Send Email Notification
stop

CONCLUSION

This project demonstrates the successful implementation of a fully automated DevOps pipeline tailored to modern software delivery needs. By combining Infrastructure as Code, configuration management, containerization, CI/CD, and monitoring, the project offers a production-ready deployment architecture for legacy applications. The result is a scalable, secure, and efficient solution that significantly improves deployment speed, reduces human error, and aligns with industry best practices. The project lays a strong foundation for future enhancements such as Kubernetes integration, serverless deployments, and cost-optimized cloud provisioning.