INT 332 – DevOps Virtualization and Configuration Management

(Online Food Donation)

Name■: Manikanta gedda

Section■■: KO016

Reg No■: 12211113 Roll No :■■63

# DevOps CI/CD Pipeline Implementation

Using Docker, Docker Swarm, React, Nodejs, Full Stack Application

Submitted To – Ritika Mahajan

## Abstract

This project presents the implementation of a CI/CD pipeline for a full-stack web application called Online Food Donation, which facilitates the donation and distribution of surplus food by connecting donors with NGOs and individuals in need. The frontend is developed using React with Vite, and the backend is built with Node.js and Express. The application is containerized using Docker and deployed using Docker Swarm. Jenkins is used to automate the CI/CD pipeline on a local Windows environment. When new code is pushed to GitHub, Jenkins pulls the latest changes, builds Docker images, pushes them to Docker Hub, and redeploys the updated containers using Docker Swarm. This DevOps pipeline ensures reliable, consistent, and efficient delivery of updates, improving productivity and reducing manual intervention during deployment.

## Introduction

In today's fast-paced development environment, delivering applications quickly and reliably is more important than ever. DevOps practices help bridge the gap between development and operations, enabling continuous delivery, better collaboration, and improved software quality. The Online Food Donation platform is a real-world full-stack application designed to connect food donors with NGOs and people in need, helping reduce food waste and serve communities more efficiently. This report highlights the DevOps pipeline built for the project to automate code integration, testing, and deployment using modern tools like Jenkins, Docker, and Docker Swarm.

## Tools & Technologies Used

Docker: Used to containerize both the backend (Node.js/Express) and the frontend (React/Vite) applications, making them portable and consistent across environments.

Docker Swarm: Managed the deployment of multiple containers, allowing the app to run in a distributed setup for better scalability and performance.

Jenkins: Acted as the core automation tool in the CI/CD pipeline. It listens for changes on GitHub, builds the updated code, and handles deployment automatically.

GitHub: Served as the version control system, storing all project code. It also triggers Jenkins builds through webhooks whenever code is pushed.

Node.js + Express: Powered the backend, handling routes and logic for food donation management, including user actions and coordination with NGOs.

React + Vite: Built a fast and responsive user interface for donors, NGOs, and volunteers. Vite was used to optimize build times during development and deployment.

Nginx: Used to serve the frontend's static files efficiently after the production build is complete.

Architecture Diagram (Textual Representation)

Flow:

Developer pushes the latest code to GitHub.

A GitHub webhook triggers a Jenkins job running on a local Windows machine.

Jenkins pulls the updated code from GitHub and builds both the backend and frontend applications.

Docker creates container images for both services and pushes them to Docker Hub.

Docker Swarm, set up on the same local environment, pulls the latest images.

The updated containers are deployed using docker stack deploy.

The application becomes accessible through the host

machine’s IP address and exposed ports.

CI/CD Workflow

Continuous Integration (CI):

Whenever a developer pushes code to the GitHub repository, a webhook automatically triggers a Jenkins job.

Jenkins pulls the latest code and begins building both the backend (Node.js/Express) and the frontend (React/Vite).

Custom Dockerfiles are used to build Docker images for both the frontend and backend.

Once built, these images are pushed to Docker Hub, ready for deployment.

Continuous Deployment (CD):

Docker Swarm pulls the latest versions of the backend and frontend images from Docker Hub.

The services are redeployed using the docker stack deploy command with a Docker Compose file.

Updated containers are started automatically with the latest code.

The application becomes accessible through the local machine’s IP address and the ports exposed by the services.

Docker & Docker Swarm

Backend Dockerfile:

The backend Dockerfile uses node:18-alpine as the base image to keep the image lightweight.

It installs all necessary dependencies and runs the Express server on port 5000.

Frontend Dockerfile:

The build stage uses node:18-alpine to compile the Vite-based React frontend into optimized static files inside the dist folder.

The production stage uses nginx:alpine to serve the contents of the dist folder, exposing the app on port 8080.

API backend Dockerfile

Client frontend Dockerfile:

Docker-Compose File:

The Docker Compose file defines two services: one for the frontend and one for the backend.

It exposes port 8080 for the frontend (served via Nginx) and port 5000 for the backend (Express server).

Both services use prebuilt Docker images stored on Docker Hub under the manikanta455 namespace.

Docker Swarm is initialized using docker swarm init on the local machine.

If needed, additional machines can be added as workers using the docker swarm join command

AWS Deployment

The application is deployed on a local Windows machine configured with Docker and Jenkins.

Docker and Jenkins were installed manually to support containerization and CI/CD automation.

The necessary ports were configured locally:

5000 for the backend (Express server),

8080 for the frontend (served by Nginx),

8081 (or similar) for Jenkins access via a browser.

The Jenkins pipeline automatically builds the frontend and backend images when changes are pushed to GitHub.

Built images are then pushed to Docker Hub, and services are redeployed using Docker Swarm with docker stack deploy.

Challenges Faced

Dockerfile path issues: Faced errors during the Vite build process due to incorrect file paths and working directories in the Dockerfile.

Port conflicts: Experienced conflicts when the same ports were already in use during local development and testing.

Jenkins file permission issues: Jenkins had limited access to certain directories and files on the Windows environment, requiring adjustments in workspace permissions.

Docker image size: Initial Docker images were large, which caused slow build and push times. This was

resolved using multi-stage builds to reduce image size and improve performance.

Docker Swarm issues: Faced challenges with services not restarting properly and occasional instability in the overlay network during the initial Swarm configuration and deployments. These were resolved through configuration tweaks and restarting the Swarm manager.

## Results / Output

The Online Food Donation app was successfully deployed using Docker Swarm on a local Windows environment.

The application is accessible locally:

Frontend: via port 8080

Backend API: via port 5000

Each code push to GitHub triggers the Jenkins pipeline automatically through a webhook.

Jenkins console output shows the full CI/CD process—from pulling the latest code, to building images, to deployment.

Docker containers for both frontend and backend are running reliably within the Swarm cluster.

Docker Hub confirms successful image pushes under:

manikanta455/frontend

manikanta455/backend

GitHub Repository: https://github.com/manikantas123/online-food- donation

## Conclusion

This project successfully implemented a production-ready CI/CD pipeline for a full-stack web application focused on food donation. By integrating Docker, Jenkins, and Docker Swarm, the deployment process was fully automated— reducing manual effort and enabling faster, more reliable updates. The use of prebuilt Docker images and a GitHub- triggered Jenkins pipeline ensured consistency across builds and environments. Going forward, the project can be improved by integrating monitoring tools like Prometheus and adding centralized logging solutions such as the ELK stack to enhance observability, error tracking, and system reliability.

## References

Docker Documentation: https://docs.docker.com/

Jenkins Documentation: https://www.jenkins.io/doc/

Vite + React: https://vitejs.dev/

Node.js: https://nodejs.org/

Nginx Documentation: https://nginx.org/en/docs/

Docker Hub: https://hub.docker.com/

GitHub: https://github.com/