TechNova DevOps Pipeline

Authors:

Dhruv Joshi

Raman Boora

Diya Tomar

Hardik Yadav

Sidharth Maalpani

Jayvardhan Singh

Here Organization:

Your Organization or Institution Name

Date:

July 3, 2025

Abstract

TechNova Solutions, a rapidly growing e-commerce startup, faced critical challenges in maintaining deployment efficiency, including manual infrastructure setup, inconsistent environments, and delayed release cycles. To address these issues, our project focuses on building a fully automated DevOps pipeline that enables consistent, scalable, and secure deployment of web applications.

The pipeline begins with a version-controlled application repository hosted on GitHub, comprising a Node.js-based dynamic web application and a static frontend designed using HTML, CSS, and JavaScript. A custom Dockerfile is used to containerize the application, ensuring consistent behavior across all environments.

We implemented a CI/CD pipeline using GitHub Actions, which automates the process of building Docker images and pushing them to Docker Hub whenever new code is committed. This ensures that every application update is packaged and ready for deployment without manual intervention. For infrastructure provisioning, we are using Terraform to automate the setup of AWS services like ECS, RDS, and IAM roles, allowing reproducible and scalable cloud deployments. To monitor performance and ensure observability, we integrated Prometheus and Grafana to track system and container-level metrics in real-time. Additionally, AWS CloudWatch is used for centralized logging and alerts.

Security is enforced through Docker image scanning using tools like Trivy, GitHub secret scanning, and secure handling of environment variables and credentials.

This project demonstrates a complete end-to-end DevOps workflow, combining containerization, continuous integration and delivery, infrastructure as code, monitoring, and security — enabling TechNova to deploy faster, scale efficiently, and maintain high application reliability.

1. Title of the Project

Technova Devops Pipeline

2. Introduction

2.1 History

Traditionally, software deployment was done manually — setting up servers, copying files, and configuring environments by hand. This often led to **inconsistencies**, **errors**, and **deployment delays**.

With growing application complexity, the industry shifted toward **DevOps practices**, which introduced automation, containerization, and continuous delivery to make deployments faster and more reliable.

Our project reflects this modern approach by implementing a **DevOps pipeline** using tools like **GitHub Actions, Docker, and Terraform**, helping automate code integration, infrastructure provisioning, and deployment processes for a web application.

2.2 Problem Statement

TechNova Solutions, a growing e-commerce startup, faces frequent deployment issues due to manual server configuration, inconsistent environments, and the absence of automated infrastructure provisioning and monitoring tools.

---

3. Objective

3.1 Main Objective

To design and implement a fully automated CI/CD pipeline for a Node.js web application using GitHub Actions, Docker, and Terraform. The goal is to provision infrastructure on AWS, containerize the application, and enable automatic deployment on every code commit.

3.2 Sub-Objectives

- Containerize a Node.js application using Docker.
- Set up GitHub Actions to automate Docker builds and push to Docker Hub.

- Write Terraform scripts to provision EC2 instances and required AWS resources.
- Integrate application deployment into the CI/CD workflow.
- Plan for CloudWatch monitoring and SonarQube static analysis integration.

---

4. System Analysis

4.1 Existing System

The existing deployment approach required manual setup of servers and hand-written shell scripts, making updates risky and non-repeatable. Developers needed to manually install dependencies, configure web servers, and restart applications—leading to inconsistent environments and high error rates.

4.2 Gaps Identified

The current system lacks automation, leading to several critical gaps:

- **Inconsistency in Environments**: Manual setup results in variations between development, testing, and production environments, causing deployment failures.
- **High Downtime and Error Rates**: The absence of standardized processes increases the risk of errors during updates, leading to prolonged downtime.
- **Limited Scalability**: Manual dependency management and server configuration hinder the ability to scale infrastructure efficiently.
- **Low Repeatability**: Hand-written scripts lack version control and documentation, making it difficult to replicate deployments reliably.
- **Developer Overhead**: Significant time is spent on repetitive manual tasks, reducing productivity and release velocity.

These gaps highlight the need for an automated, standardized, and scalable deployment process to address the inefficiencies of the existing system.

4.3 Motivation

To minimize downtime, reduce manual overhead, and standardize environments, the team decided to automate the entire deployment cycle—from code push to production deployment—using modern DevOps tools. This approach improves release velocity, consistency, and team productivity.

4.4 Proposed System

The proposed system introduces:

- **GitHub** – For version control and workflow automation, enabling collaborative development and traceable code changes.
- **Docker** – To ensure environment consistency by packaging the application and its dependencies into containers.
- **Terraform** – For automated and repeatable infrastructure provisioning on AWS.
- **GitHub Actions** – To automate the CI/CD pipeline, from code push to Docker image creation and deployment.
- **AWS EC2** – As the deployment platform, providing scalable and reliable infrastructure for running the application.
- **Amazon CloudWatch** – For monitoring system performance, logging, and early detection of issues in deployed services.
- **SonarQube** – For static code analysis to identify bugs, vulnerabilities, and maintain code quality during development.

---

5. Design

5.1 UI/UX Design

A basic Node.js web application serves the frontend of our website, featuring a welcome page with navigation to 2–3 additional pages. While simple, it effectively demonstrates multi-page routing and supports our CI/CD pipeline testing and deployment.

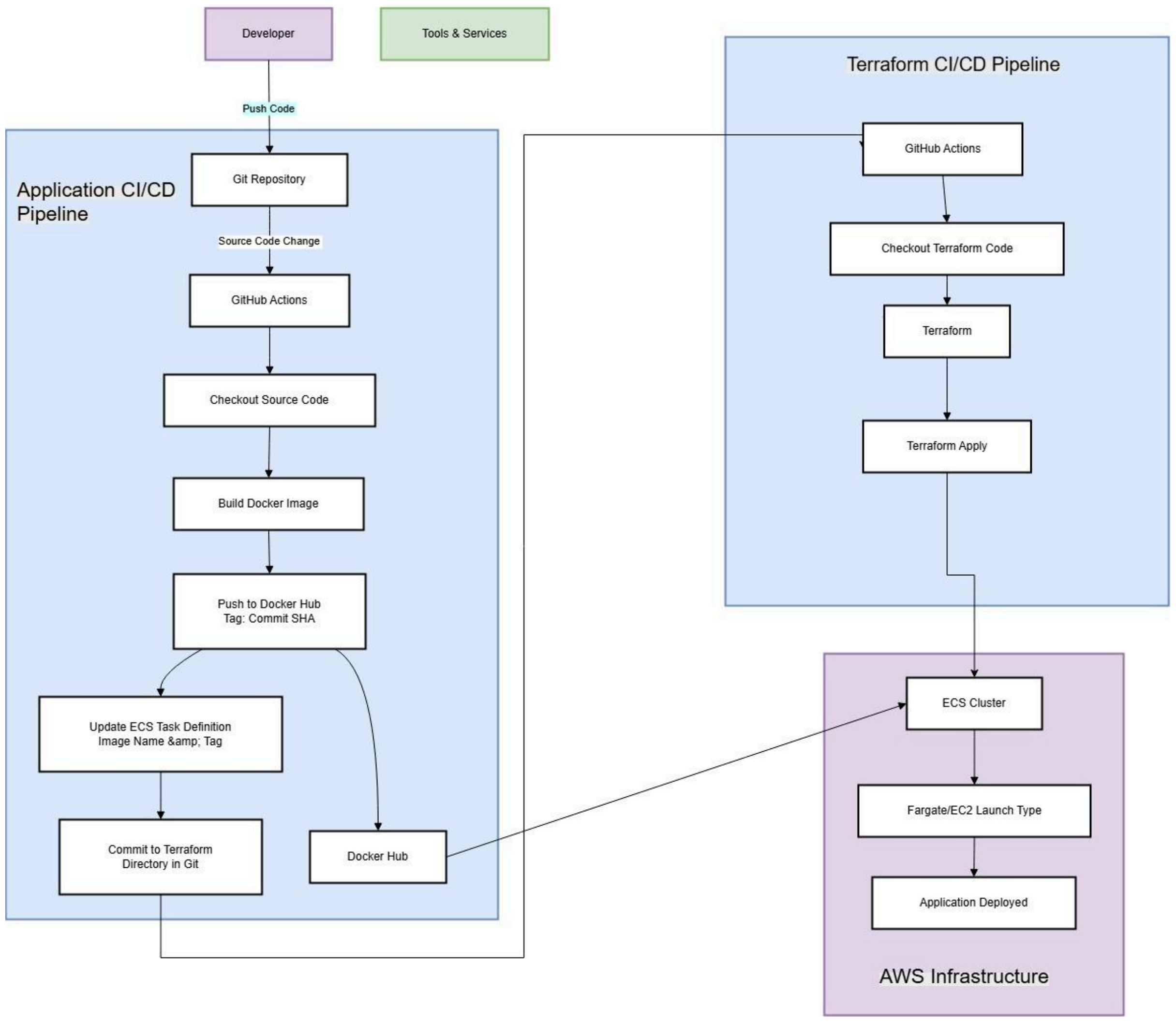
5.2 Use Case Flow

- **Developer pushes code to GitHub**
  The latest changes to the application are committed and pushed to the main branch of the repository.
- **GitHub Actions build and push a Docker image**
  A CI/CD workflow is triggered automatically, which builds the Docker image and

pushes it to Docker Hub.

- **Terraform provisions AWS EC2 infrastructure**
  Using Infrastructure as Code, Terraform creates and configures EC2 instances, networking, and security groups.

- **GitHub Actions deploy the container to EC2**
  The workflow pulls the Docker image from Docker Hub and runs the container on the provisioned EC2 instance.



---

6. Technical Architecture

6.1 Frontend Layer

The Node.js app serves a static HTML page:

- Built with Express.js
- Styled with plain CSS for simplicity
- Demonstrates runtime web hosting within Docker

6.2 CI/CD Layer

- **GitHub Actions** automates:
  - Code checkout
  - Docker build
  - Push to Docker Hub
- **Secrets** (Docker credentials) are managed securely in GitHub

6.3 Containerization Layer

- **Dockerfile** packages the Node.js app
- Containers run identically in all environments

6.4 Infrastructure Layer

- **Terraform** provisions:
  - AWS EC2 instance
  - Security Groups
  - IAM roles (planned)
- Resources are reproducible and version-controlled

---

7. Implementation

7.1 Dockerization

- Dockerfile defines how to build and run the app
- Tested locally and in CI to ensure consistency

7.2 GitHub Actions Workflow

- Triggered on main branch push
- Builds Docker image
- Pushes to Docker Hub with SHA tag

7.3 Terraform Infrastructure Pipeline

- Terraform script defines AWS infrastructure
- Terraform GitHub Actions pipeline (implemented) automates provisioning

7.4 Work in Progress

- ECS-based deployment to be connected
- CloudWatch logs and monitoring integration pending
- SonarQube for code quality planned in CI

---

8. Limitations and Future Enhancements

Current Limitations

- **CloudWatch integration pending**: Centralized logging and real-time monitoring using AWS CloudWatch has not yet been fully implemented.

- **SonarQube setup incomplete**: Static code analysis for identifying bugs, vulnerabilities, and code smells is not yet integrated into the CI pipeline.

- **ECS deployment not fully automated**: While infrastructure provisioning with Terraform is in place, ECS task definitions and service configurations are still being finalized.

- **Extra functionality (e.g., chatbot)** is yet to be implemented: A planned feature such as a chatbot or similar interactive component is pending and will be added in the final phase.

Planned Enhancements

- **Automate ECS Service Deployment**
  Implement full automation of ECS task definitions and docker image updates using GitHub Actions.

- **Integrate AWS CloudWatch Monitoring**
  Stream container and application logs to CloudWatch, and set up alarms for service failures or abnormal behavior.

- **Add SonarQube to CI Pipeline**
  Integrate SonarQube scanning into the GitHub Actions workflow to enforce code quality standards and detect vulnerabilities early.

- **Add Trivy to CI Pipeline**
  Integrate Trivy Docker image scanning into the GitHub Actions CI pipeline to identify vulnerabilities early and enhance container security.

- **Enhance the Node.js Application**
  Extend the app to include backend features such as database connectivity (e.g., MongoDB/MySQL) and user authentication.

---

9. Team Contribution

The development of the TechNova DevOps Pipeline was a collaborative effort, with each team member taking ownership of specific components to ensure a well-structured and end-to-end automated deployment system. The contributions are as follows:

- **Jayvardhan Singh**
  Took responsibility for designing and implementing the **Dockerfile** used to containerize the Node.js application. He also developed the **frontend interface**, ensuring a functional and responsive webpage was available for deployment and testing.

- **Diya Tomar & Dhruv Joshi**
  Collaboratively built and implemented the **CI/CD pipeline** using GitHub Actions, automating the build and deployment workflow. They also designed and integrated a **basic chatbot interface** into the frontend, showcasing how interactive features can be deployed through the pipeline.

- **Hardik Yadav**
  Led the implementation of **Application Load Balancer (ALB)** setup through Terraform and researched integration of **SonarQube** into the CI pipeline for static code analysis and quality checks.

- **Sidharth Maalpani**
  Focused on networking and access control, including setting up the **VPC (Virtual Private Cloud)**, configuring **Security Groups**, and assigning necessary **IAM roles** to ensure secure communication and deployment permissions.

- **Raman Boora**
  Managed the setup of the **ECS Cluster**, including the definition of task and service configurations. He also handled **Blue-Green deployment strategy** planning and configured the **Terraform remote state backend** for team-based infrastructure collaboration.

---

8. Conclusion

This project successfully established the foundation for a robust and automated DevOps pipeline. The Node.js application has been fully containerized using Docker, and the CI/CD process—powered by GitHub Actions—automatically builds and pushes Docker images with every code commit.

Terraform scripts have been implemented to provision AWS infrastructure reliably, eliminating the need for error-prone manual setup.

The next phase will focus on deploying the application to AWS ECS, integrating monitoring through CloudWatch, and enforcing code quality checks using SonarQube.

Once complete, TechNova will benefit from a scalable, reliable, and production-ready deployment pipeline—allowing faster releases, improved system visibility, and reduced operational overhead.