RetailMax: E-Commerce Monolith Cloud Migration Platform

Team Members

Aditya Tomar
Ansh Tyagi
Ritesh Murari
Ujjwal Chauhan
Yogesh Gaur

July 2, 2025

Abstract

RetailMax, a legacy e-commerce platform, faced deployment inefficiencies and frequent downtimes due to its monolithic architecture. This project aims to automate RetailMax's infrastructure and deployment pipeline through DevOps practices, focusing on cloud migration, containerization, CI/CD automation, configuration management, and real-time monitoring. By leveraging modern tools like Docker, Jenkins, Terraform, Ansible, and Prometheus/Grafana, the system now supports scalable, zero-downtime deployments with improved visibility and faster feature rollouts.

Table of Contents

1. Abstract
2. Introduction
3. Problem Statement
4. Objectives
5. Technology Stack
6. Methodology
   - Infrastructure Provisioning
   - Containerization
   - CI/CD Pipeline
   - Configuration Management
   - Monitoring & Observability
7. Implementation Steps
8. Challenges & Solutions
9. Conclusion
10. Future Enhancements
11. References
12. Appendix

Introduction

RetailMax is an established e-commerce company operating a monolithic web application. However, as user demand and product complexity grew, the traditional deployment model proved inefficient. Manual server configurations, inconsistent environments, and frequent downtimes during updates hindered performance and customer experience. Recognizing the need for digital transformation, this project aims to modernize RetailMax's infrastructure using DevOps and cloud-native solutions.

Problem Statement

The existing monolithic architecture at RetailMax creates several bottlenecks:

- Manual deployments cause high risk of human error
- Feature updates lead to unplanned downtime
- Lack of scalability to handle increased traffic during sales
- No unified monitoring or performance visibility

To resolve this, we propose an automated solution that enables reliable, scalable, and observable deployments using containerization, CI/CD, and cloud provisioning.

Objectives

1. Re-architect the monolith for container-based deployment
2. Automate the build, test, and deployment pipeline using Jenkins
3. Use Terraform to provision consistent AWS infrastructure
4. Apply Ansible for configuration management and deployment automation
5. Integrate Prometheus and Grafana for system monitoring and alerting

Technology Stack

Tool	Purpose
GitHub	Version control
Docker	Containerization of the monolith
Jenkins	CI/CD pipeline automation
AWS EC2/ECS	Cloud hosting and scalability
Terraform	Infrastructure as Code (IaC)
Ansible	Configuration and deployment automation
Prometheus/Grafana	Monitoring and visualization

Methodology

Infrastructure Provisioning

We used **Terraform** to define and provision cloud infrastructure on AWS. The configuration includes:

- Creating a custom VPC with public/private subnets
- Setting up EC2 instances for deployment
- Defining security groups and access policies
- Automating key provisioning and resource linking

This enables consistent, versioned, and repeatable infrastructure setup with minimal manual effort.

Containerization

The legacy application was containerized using **Docker**. A custom Dockerfile was written to:

- Install required dependencies
- Copy source code and configuration files
- Expose necessary ports and run the app

This ensures portability and consistency across development, staging, and production environments.

CI/CD Pipeline

A **Jenkins pipeline** was created with the following stages:

- **Clone Repository**: Pull latest code from GitHub
- **Build Docker Image**: Use Dockerfile to build the image
- **Push to Docker Hub**: Store the image for deployment
- **Deploy to EC2**: Trigger Ansible playbook to deploy the container

The pipeline ensures automated, repeatable, and fast releases with minimal human intervention.

Configuration Management

Using **Ansible**, we automated:

- Software installation (e.g., Docker, Node.js, databases)
- Docker setup and container run commands
- Environment variable configuration

Playbooks were modular and versioned, supporting rollback and scalability.

Monitoring & Observability

To monitor application performance and resource usage, we:

- Installed **Prometheus** for metrics collection
- Integrated **Grafana** for visualization dashboards
- Configured alerts for CPU, memory, and traffic spikes

This helped ensure visibility into the app's health, enabling quick resolution of issues.

Implementation Steps

1. Initialize Terraform and provision AWS infrastructure
2. Containerize application with Docker
3. Set up Jenkins pipeline connected to GitHub
4. Create Docker image and push to Docker Hub
5. Configure EC2 servers with Ansible playbooks
6. Deploy Docker container via Ansible
7. Install Prometheus and Grafana
8. Create dashboards for monitoring
9. Test deployments and verify monitoring alerts

Challenges & Solutions

Challenge	Solution
Terraform errors due to AWS permissions	Created IAM roles and policy attachments
Docker container failed on EC2	Fixed port mappings and dependencies in Dockerfile
Jenkins agent connectivity issues	Configured SSH keys and EC2 instance firewall
Playbook idempotency in Ansible	Used handlers and tags to control re-execution
Prometheus targets not showing	Verified scrape configs and firewall rules

Conclusion

The RetailMax project successfully modernized a legacy monolith using DevOps principles. Through automation, containerization, and cloud-native tooling, we eliminated manual deployment bottlenecks, reduced downtime, and achieved faster release cycles. The infrastructure is now scalable, observable, and production-ready.

Future Enhancements

- Break monolith into microservices using Kubernetes
- Use AWS ECS/Fargate for fully managed container orchestration
- Implement cost optimization using AWS budgets and tagging

References

1.Terraform Documentation

https://developer.hashicorp.com/terraform/docs

2.Docker Docs

https://docs.docker.com

3.Jenkins Pipelines

https://www.jenkins.io/doc/book/pipeline/

4.Ansible Playbooks

https://docs.ansible.com/ansible/latest/user_guide/playbooks.html

5.Prometheus Monitoring

https://prometheus.io/docs/introduction/overview/

6.Grafana Dashboards

https://grafana.com/docs/grafana/latest/dashboards/

Appendix

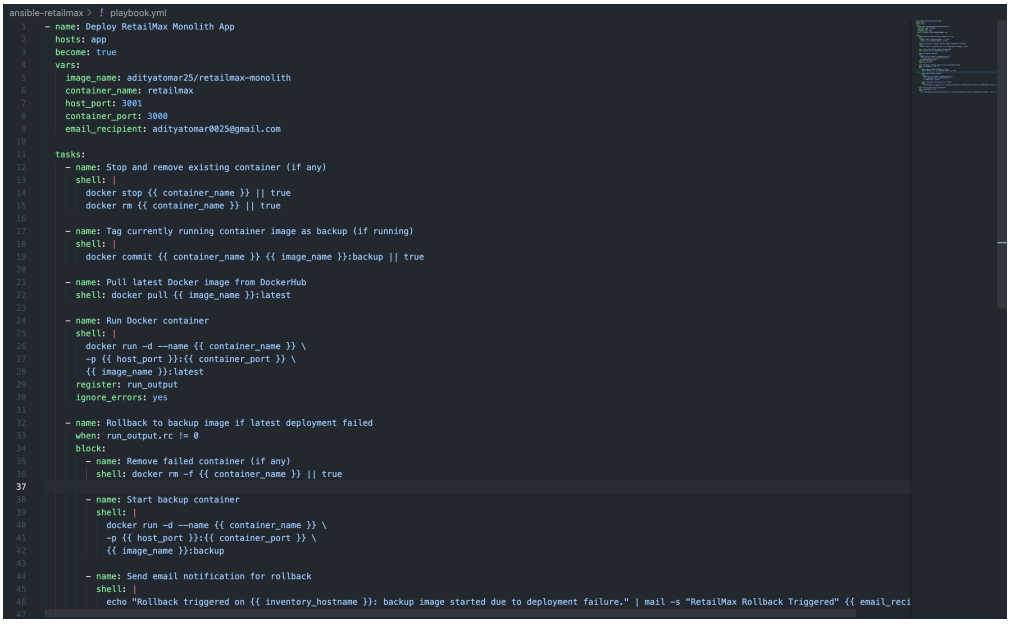
Terraform

Dockerfile

Jenkinsfile

```
monolith-app > Jenkinsfile
pipeline {
    agent any

    environment {
        IMAGE_NAME = 'adityatomar25/retailmax-monolith'
        IMAGE_TAG = "build-${BUILD_NUMBER}"
    }

    stages {
        stage('Checkout Code') {
            steps {
                git branch: 'main',
                    credentialsId: 'b2c3e09b-26e7-4572-b583-c4cbd70387c4',
                    url: 'https://github.com/adityatomar25/RetailMax.git'
            }
        }
        stage('Debug Workspace') {
            steps {
                sh 'ls -R'
    }
}

        stage('Install Dependencies') {
            steps {
                dir('monolith-app/backend') {
                    sh 'npm install'
                }
            }
        }

        stage('Run Unit Tests') {
            steps {
                dir('monolith-app/backend') {
                    sh 'npm test'
                }
            }
        }

        stage('Build Docker Image') {
            steps {
                dir('monolith-app') {
                    sh "docker build -t ${IMAGE_NAME}:${IMAGE_TAG} ."
                }
            }
        }

        stage('Scan Docker Image (Trivy)') {
```

Ansible



```
ansible-retailmax > ! playbook.yml
- name: Deploy RetailMax Monolith App
  hosts: app
  become: true
  vars:
    image_name: adityatomar25/retailmax-monolith
    container_name: retailmax
    host_port: 3001
    container_port: 3000
    email_recipient: adityatomar0025@gmail.com

  tasks:
    - name: Stop and remove existing container (if any)
      shell: |
        docker stop {{ container_name }} || true
        docker rm {{ container_name }} || true

    - name: Tag currently running container image as backup (if running)
      shell: |
        docker commit {{ container_name }} {{ image_name }}:backup || true

    - name: Pull latest Docker image from DockerHub
      shell: docker pull {{ image_name }}:latest

    - name: Run Docker container
      shell: |
        docker run -d --name {{ container_name }} \
        -p {{ host_port }}:{{ container_port }} \
        {{ image_name }}:latest
      register: run_output
      ignore_errors: yes

    - name: Rollback to backup image if latest deployment failed
      when: run_output.rc != 0
      block:
        - name: Remove failed container (if any)
          shell: docker rm -f {{ container_name }} || true

        - name: Start backup container
          shell: |
            docker run -d --name {{ container_name }} \
            -p {{ host_port }}:{{ container_port }} \
            {{ image_name }}:backup

        - name: Send email notification for rollback
          shell: |
            echo "Rollback triggered on {{ inventory_hostname }}: backup image started due to deployment failure." | mail -s "RetailMax Rollback Triggered" {{ email_reci
```