TechNova Solutions Reference Architecture Document

# Executive Summary

TechNova Solutions, a fast-growing e-commerce startup, requires a fully automated DevOps pipeline. This architecture ensures reliable provisioning and configuration of cloud infrastructure, automated build and testing, secure artifact packaging, and hands-free deployments to production. The goal is to deliver consistent, version-controlled environments, faster delivery cycles, reduced failures, and easy scalability during peak traffic periods.

# Architecture Objectives

1. Fully automated CI/CD pipeline (provision, build, test, package, deploy)  
2. Ensure consistent, version-controlled environments  
3. Enable faster delivery cycles  
4. Reduce deployment failures  
5. Support easy scalability for peak traffic periods

# Proposed Tech Stack

|  |  |
| --- | --- |
| Component | Technology |
| Version Control | GitHub |
| CI/CD Orchestration | Jenkins |
| Containerization | Docker |
| Infrastructure Provisioning | Terraform + Ansible |
| Deployment Platform | AWS ECS/EC2 + ALB |
| Database | AWS RDS (PostgreSQL) |
| Artifact Storage | AWS ECR |

# Reference Architecture Diagram

A diagram of a cloud computing system

AI-generated content may be incorrect.

# Pipeline Workflow

1. Developer pushes code to GitHub  
2. Jenkins pipeline triggers automatically  
3. Terraform provisions AWS infrastructure  
4. Docker builds application container  
5. Automated tests validate functionality  
6. Docker image pushed to AWS ECR  
7. Jenkins deploys container to ECS cluster  
8. Load balancer routes user traffic

# Requirement Mapping

|  |  |
| --- | --- |
| Requirement | Solution |
| Fully Automated Pipeline | Jenkins + Terraform + Docker orchestrates provisioning, testing, deployment |
| Consistent, Version-Controlled Environments | GitHub + IaC + Docker ensure reproducibility |
| Faster Delivery Cycles | Automated CI/CD pipeline |
| Reduced Deployment Failures | Automated testing & health checks |
| Easy Scalability | AWS ECS with ALB and auto scaling |

# TechNova Pipeline Workflow (Detailed Explanation)

# 1. Developer pushes code to GitHub

# A TechNova developer commits new code (bug fix, UI update, new feature) to the GitHub repository.

# This repository stores:

# /Webapp → Webapplication source code

# /infra → Terraform & Ansible scripts

# /docker → Dockerfile & configs

# A webhook is configured so GitHub automatically notifies Jenkins whenever code is pushed or a pull request is merged.

# 👉 Benefit: Kicks off automation immediately after code changes, reducing delays.

# 2. Jenkins pipeline triggers automatically

# Jenkins, the CI/CD orchestrator, listens for GitHub triggers.

# A Jenkins pipeline (defined in a Jenkinsfile) begins execution.

# Jenkins pulls the latest code from the repository.

# 👉 Benefit: No manual intervention — every code change is tested & deployed automatically.

# 3. Terraform provisions AWS infrastructure

# Jenkins executes Terraform scripts from the /infra/terraform folder.

# Terraform ensures the required AWS resources exist or are updated:

# ECS Cluster for container hosting

# RDS Database for e‑commerce data

# Application Load Balancer (ALB) for traffic routing

# Auto Scaling Groups to handle traffic spikes

# If infra already exists, Terraform only applies necessary updates.

# 👉 Benefit: Infrastructure is consistent, reproducible, and version‑controlled.

# 4. Docker builds application container

# Jenkins uses the /docker/Dockerfile to package the TechNova application into a Docker image.

# The Dockerfile ensures:

# Correct OS & runtime (Node.js / Python Flask).

# All dependencies installed (e.g., NPM packages, pip modules).

# Application entry point defined (npm start / python app.py).

# 👉 Benefit: App runs the same way in dev, test, and prod environments.

# 5. Automated tests validate functionality

Jenkins runs test scripts inside the Docker container:

Unit tests (check individual functions).

Integration tests (e.g., checkout flow with database).

API endpoint tests (verify responses & status codes).

If any test fails → pipeline stops and notifies the developer.

# 👉 Benefit: Prevents broken code from reaching production.

# 6. Docker image pushed to AWS ECR

On test success, Jenkins tags the Docker image with a version (e.g., v1.5.2).

The image is pushed to AWS Elastic Container Registry (ECR).

Stored images allow rollback if a future deployment fails.

# 👉 Benefit: Versioned artifacts ensure safe rollbacks and auditability.

# 7. Jenkins deploys container to ECS cluster

Jenkins updates the AWS ECS Service to use the new Docker image.

ECS pulls the image from ECR and spins up new containers.

Old containers are gracefully shut down after new ones pass health checks.

(This is a rolling deployment).

# 👉 Benefit: Deployment is seamless with zero downtime.

# 8. Load balancer routes user traffic

The Application Load Balancer (ALB) distributes traffic across multiple ECS tasks.

ALB health checks ensure only healthy containers receive traffic.

Auto Scaling rules can add more ECS tasks during high traffic (e.g., festive sales).

# 👉 Benefit: High availability and performance during peak loads.

# Conclusion

This reference architecture enables TechNova to move from manual deployments to a fully automated DevOps framework. By leveraging CI/CD automation, Infrastructure as Code, containerization, and cloud scalability, TechNova can achieve faster delivery cycles, reduced risk of failure, and seamless scalability during peak periods.

# Version history

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Sno | Date | Version number | Owner name | Contributor name | State |
| 1 | 31-07-25 | 1.0 | Vaibhav | Simran  Bhumi  Yaksha  Akshat | Draft |
|  |  |  |  |  |  |
|  |  |  |  |  |  |