# 4. System Analysis & Design

## 1. Problem Statement & Objectives

Problem Statement:  
Manual deployment processes are prone to errors and delays, impacting software delivery and operational efficiency. The objective of this project is to implement a fully automated DevOps pipeline using Git, Jenkins, Maven, Docker, Kubernetes, terraform, Ansible, and AWS to enhance build, test, and deployment processes.  
  
Project Goals:  
- Streamline CI/CD processes to minimize manual intervention.  
- Implement robust containerization and orchestration for scalability.  
- Achieve high availability and quick recovery from failures using Kubernetes and AWS.

## Use Case Diagram & Descriptions

System Actors:  
- Developer: Commits code and triggers CI/CD pipeline.  
- DevOps Engineer: Manages pipeline configurations and deployments.  
- QA Tester: Reviews automated test results.  
- End User: Interacts with the deployed application.  
System Interactions:  
- Code commit triggers automated build and test processes.  
- Successful builds are containerized and deployed to Kubernetes on AWS.  
- Monitoring tools provide real-time system performance metrics.  
A diagram of a company

Description automatically generated

A diagram of a company

Description automatically generated

A diagram of a company

Description automatically generated

## Functional & Non-Functional Requirements

Functional Requirements:  
- Version control with Git.  
- CI/CD pipeline automation with Jenkins.  
- Build automation using Maven.  
- Application containerization with Docker.  
- Orchestration with Kubernetes.  
- Cloud hosting on AWS.  
  
Non-Functional Requirements:  
- Performance: Maintain response time under 2 seconds.  
- Security: Implement role-based access and secure communication.  
- Usability: User-friendly logs and monitoring dashboards.  
- Reliability: Achieve 99.9% uptime with Kubernetes' self-healing features.  
- Scalability: Auto-scaling application instances based on demand.

## Software Architecture

High-Level Design:  
- Architecture Style: Microservices architecture managed by Kubernetes.  
- Components: Jenkins (CI/CD), Docker (Containerization), Kubernetes (Orchestration), AWS (Cloud Hosting).  
- Interactions: Automated build, test, and deployment pipelines with integrated monitoring and alerting.  
- Accessibility: Ensure the interface is accessible to all user groups.

## System Deployment & Integration

Technology Stack:  
- Backend: flask, Golang , Frontend: node.js (for any user-facing interfaces), Database: MySQL .  
  
Deployment Diagram:  
- Describes how software components are distributed across hardware, including Kubernetes nodes and AWS services.  
  
Component Diagram:  
- Shows high-level system components, their roles, and dependencies.