

**TRIBHUVAN UNIVERSITY**

**FACULTY OF HUMANITIES AND SOCIAL SCIENCE**

**DevOps Butler**

**A PROJECT REPORT**

**SUBMITTED TO**

**Department of Computer Application**

**Reliance College, Kathmandu**

***In partial fulfillment of the requirements for the Bachelors in Computer Application***

**SUBMITTED BY**

Aayush Adhikari (Reg No: - 6-2-522-172-2020)

July,2025

**Under the Supervision of:**

Abhijeet Kumar Sah



**TRIBHUVAN UNIVERSITY**

**FACULTY OF HUMANITIES AND SOCIAL SCIENCE**

**Reliance College**

**Supervisor’s Recommendation**

I hereby recommend that this project prepared under my supervision by Aayush Adhikari entitled **DevOps Butler** in particular fulfillment of the requirements for the degree of Bachelor of Computer Application is recommended for the final evaluation.

….………………..

**SIGNATURE**

Abhijeet Kumar Sah

**Department of Computer Application**

Chabahil, Saraswatinagar, Kathmandu



**TRIBHUVAN UNIVERSITY**

**FACULTY OF HUMANITIES AND SOCIAL SCIENCE**

**Reliance College**

**LETTER OF APPROVAL**

This is to certify that this project prepared by Aayush Adhikari entitled **DevOps Butler** in particular fulfillment of the requirements for the degree of Bachelor in Computer Application has been evaluated. In our opinion it is satisfactory in the scope and quality as a project for the required degree.

|  |  |
| --- | --- |
| **….……………………**  Abhijeet Kumar Sah  **Supervisor**  Saraswatinagar, Kathmandu | **….……………………**  Aayush Dhital  **Coordinator**  Saraswatinagar, Kathmandu |
| **….……………………**  **Internal Examiner** | **….……………………**  **External Examiner** |

# ABSTRACT

DevOps Butler is an AI-powered, developer-centric deployment automation platform built to simplify and accelerate the local deployment of containerized applications from any Git repository. Designed with full-stack projects in mind, it intelligently detects Docker or Docker Compose configurations—or generates them when absent—leveraging best practices for security, scalability, and performance. The platform features a modern, intuitive web interface that allows users to securely upload .env files, configure services, and receive real-time deployment feedback. By integrating Docker, Docker Compose, and Nginx with intelligent analysis, DevOps Butler resolves common issues like port conflicts, missing files, and networking challenges. It ensures environment isolation, automatically cleans up unused resources, and enforces strong security standards by never storing sensitive environment data and actively warning users against exposing secrets. Whether onboarding new developers or testing production-ready stacks, DevOps Butler transforms the local deployment experience into a seamless, automated, and secure process—enabling teams to focus on building, not troubleshooting.

# ACKNOWLEDGEMENT

I would like to extend my heartfelt gratitude to my project supervisor, Mr. Abhijeet Kumar Sah, whose expert guidance, consistent support, and insightful feedback were instrumental throughout the development of DevOps Butler. His mentorship not only helped shape the technical foundation of this project but also instilled in me the discipline and clarity needed to complete it with confidence.

My sincere appreciation also goes to my classmates at Reliance College, whose collaboration, encouragement, and shared learning moments created an environment of growth and motivation. I am equally thankful to Reliance College for equipping us with the tools, infrastructure, and opportunities that enabled us to explore and innovate fearlessly. To my family—your unwavering belief in me has been my anchor. Your support gave me the resilience to face every challenge head-on. Being part of this journey has been a deeply transformative experience. I am grateful to all who contributed to the realization of this project and to the community that constantly pushed me toward excellence.

Aayush Adhikari

DevOps Butler – AI-powered Deployment Automation Platform

**Table of Contents**

[ABSTRACT i](#_Toc203857672)

[ACKNOWLEDGEMENT ii](#_Toc203857673)

[List of Figures v](#_Toc203857674)

[List of Tables vi](#_Toc203857675)

[List of Abbreviations vii](#_Toc203857676)

[CHAPTER 1. INTRODUCTION 1](#_Toc203857677)

[1.1. Introduction 1](#_Toc203857678)

[1.2. Problem Statement 1](#_Toc203857679)

[1.3. Objectives 2](#_Toc203857680)

[1.4. Scope and limitations 2](#_Toc203857681)

[1.5. Development Methodology 3](#_Toc203857682)

[1.6. Report Organization 4](#_Toc203857683)

[CHAPTER 2: BACKGROUND STUDY AND LITERATURE REVIEW 6](#_Toc203857684)

[2.1. Background Study 6](#_Toc203857685)

[2.2. Literature Review 7](#_Toc203857686)

[CHAPTER 3: SYSTEM ANALYSIS AND DESIGN 9](#_Toc203857687)

[3.1. System Analysis 9](#_Toc203857688)

[3.1.1. Requirement Analysis 9](#_Toc203857689)

[3.1.2. Feasibility Analysis 12](#_Toc203857690)

[3.1.3. Object Modelling using Class and Object Diagrams 14](#_Toc203857691)

[3.1.4. Dynamic Modelling using State and Sequence Diagrams 15](#_Toc203857692)

[3.1.5. Dynamic Modelling using State and Sequence Diagrams 15](#_Toc203857693)

[3.2. System Design 16](#_Toc203857694)

[3.2.1. Refinement of Class, Object, State, Sequence, and Activity Diagrams 16](#_Toc203857695)

[3.2.2. Component Diagrams 17](#_Toc203857696)

[3.2.3. Deployment Diagrams 17](#_Toc203857697)

[CHAPTER 4: IMPLEMENTATION AND TESTING 19](#_Toc203857698)

[4.1. Implementation 19](#_Toc203857699)

[4.1.1. Tools Used 19](#_Toc203857700)

[4.1.2. Algorithms 20](#_Toc203857701)

[4.1.3. Implementation of modules 22](#_Toc203857702)

[4.2. Testing 23](#_Toc203857703)

[4.2.1. Unit Testing 23](#_Toc203857704)

[4.2.2 System Testing 24](#_Toc203857705)

[CHAPTER 5: CONCLUSION AND FUTURE RECOMMENDATIONS 25](#_Toc203857706)

[5.1. Conclusion 25](#_Toc203857707)

[5.2. Future Recommendations 25](#_Toc203857708)

[REFERENCES 27](#_Toc203857709)

# List of Figures

[**Figure 1.1: Agile Methodology** 4](#_Toc203859512)

**Figure 3.1: Use Case Diagram………………………………………………………….11**

[**Figure 3.3: Class Diagram** 15](#_Toc203859513)

# List of Tables

**Table 3.1: Gantt Chart…………………………………………………………………………..14**

**Table 4.1: Successful deployment……….……………………………………………..23**

[**Table 4.1: System Testing** 24](#_Toc203859332)

# List of Abbreviations

|  |  |
| --- | --- |
| AI | Artificial Intelligence |
| API | Application Programming Interface |
| CSS | Cascading Style Sheets |
| CDN | Content Delivery Network |
| Nginx | Engine X (Web Server & Reverse Proxy) |
| CRUD | Create, Read, Update, Delete |
| DB | Database |
| GPU | Graphics Processing Unit |
| HTTP | HyperText Transfer Protocol |
| HTML | HyperText Markup Language |
| IDE | Integrated Development Environment |
| JPG / JPEG | Joint Photographic Experts Group |
| UML | Unified Modeling Language |
| PNG | Portable Network Graphics |
| ML | Machine Learning |
| REST | Representational State Transfer |
| JSON | JavaScript Object Notation |
| OS | Operating System |
| MVC | Model-View-Controller |
| SQL | Structured Query Language |
| Ngrok | Next Generation Reverse Proxy |
| Flask | A Python Micro Web Framework |
| UI | User Interface |
| UX | User Experience |
| VS Code | Visual Studio Code |

# CHAPTER 1. INTRODUCTION

## 1.1. Introduction

The increasing complexity of modern software development has heightened the need for efficient, secure, and intelligent deployment tools. As teams adopt containerized architectures and microservices, local deployment environments often become bottlenecks—plagued by port conflicts, misconfigurations, and onboarding friction. DevOps Butler addresses this challenge by offering a smart, AI-assisted local deployment automation platform that simplifies the process of launching full-stack applications directly from any Git repository. Built as a final-year project, DevOps Butler integrates technologies like Docker, Docker Compose, and Nginx to manage the orchestration and networking of containerized services. What sets it apart is its AI-driven backend, which analyzes the project structure and automatically generates or patches Dockerfiles and Compose configurations. The platform features a modern web-based frontend where users can input Git URLs, upload .env files for secure environment configuration, and monitor real-time deployment logs via WebSockets. The backend, developed using FastAPI, manages the entire pipeline—from repository cloning and structure analysis to container orchestration and reverse proxy configuration. Nginx is containerized to dynamically serve deployed applications at clean, accessible URLs (e.g., project.localhost). Sensitive data like .env files are never stored, and the system enforces best practices by alerting users if secrets are committed in code. Cleanup processes ensure no orphaned containers or configs persist after project teardown.

DevOps Butler demonstrates how AI can streamline DevOps workflows while maintaining high standards of security and user experience. It is designed to make local development faster, smarter, and safer—empowering teams to focus on building, not debugging deployment pipelines.

## 1.2. Problem Statement

Modern software development has embraced containerization as the standard for building scalable and portable applications. While platforms like Docker and orchestration tools such as Docker Compose and Kubernetes have greatly enhanced deployment flexibility, the initial setup and configuration required to get a project running locally remains a pain point—especially for full-stack applications pulled from Git repositories. Developers often face issues like missing or misconfigured Dockerfiles, inconsistent environment variable handling, port conflicts, and tangled network setups. These obstacles not only slow down development but also hinder onboarding for new team members, who must often spend hours just to replicate the correct local environment. Despite the abundance of DevOps tools for production environments, there is a noticeable gap in user-friendly, intelligent platforms that focus specifically on local development automation. Most existing solutions require manual configuration, prior DevOps knowledge, or are overly rigid, making them unsuitable for early-stage developers or dynamic project structures. Moreover, managing secrets such as .env files is often overlooked or handled insecurely—leading to potential leaks or bad practices like hardcoding credentials or committing them to version control.

DevOps Butler was conceived to solve this gap. It tackles the friction in local deployment by automating the most error-prone and time-consuming parts of the setup process. By integrating AI for intelligent analysis and configuration generation, it eliminates the need for developers to manually write or debug deployment files. It also enforces security standards around sensitive data and provides a clean, responsive interface for interacting with deployments. The goal is to remove the DevOps overhead from local development, allowing teams to spin up full-stack environments from any Git repository within minutes—securely, reliably, and without expert-level intervention.

## 1.3. Objectives

The primary objectives of the **DevOps Butler – Deployment Automation Platform** are:

a) Develop a backend system that automates local deployment of full-stack applications using Docker, Docker Compose, and Nginx.

b) Integrate AI models like Gemini or OpenAI to analyze project structures and generate or patch Dockerfiles and Compose configurations.

c) Implement secure handling of .env files to ensure secrets are never stored or exposed during deployment.

d) Provide a responsive web interface for submitting Git URLs, uploading environment files, and monitoring real-time deployment logs.

e) Enable dynamic Nginx configuration to serve deployed projects at clean, isolated local URLs.

## 1.4. Scope and limitations

**1.4.1 Scopes:**

DevOps Butler focuses on providing an AI-powered, secure, and user-friendly platform for automating the local deployment of containerized applications. The current implementation covers the following:

**• Git-Based Deployment:**

Allows users to deploy full-stack applications directly from public Git repositories with minimal configuration.

**• AI-Powered Configuration Generation:**

Utilizes AI models like Gemini or OpenAI to analyze project structure and auto-generate or patch Dockerfiles and Compose files.

**• Web-Based Interface:**

Accessible via any modern browser, the frontend provides Git URL submission, .env file upload, and real-time deployment logs via WebSockets.

**• Secure .env Handling:**

Supports frontend and backend .env uploads without storing them on the server or database, ensuring secret isolation.

**• Dynamic Reverse Proxy:**

Uses a containerized Nginx proxy to expose deployed apps at clean, local URLs (e.g., http://project.localhost).

**• Real-Time Feedback & Logs:**

Displays live container logs and deployment statuses for user transparency during setup and runtime.

**1.4.2 Limitations:**

While DevOps Butler offers a robust framework for automated local deployments, the current version has some limitations:

**• No Private Repo Support Yet:**

Only public Git repositories are supported; integration with GitHub OAuth or deploy keys is pending.

**• Limited-Service Templates:**

Currently supports common stacks like Node.js, React, Django, and Next.js; advanced multi-service templates are under development.

**• Manual AI Triggering:**

AI intervention for file generation is automatic but lacks manual override or prompt customization by the user.

**• No Persistent Container State:**

All deployments are session-based and not designed for long-term persistence across reboots or shutdowns.

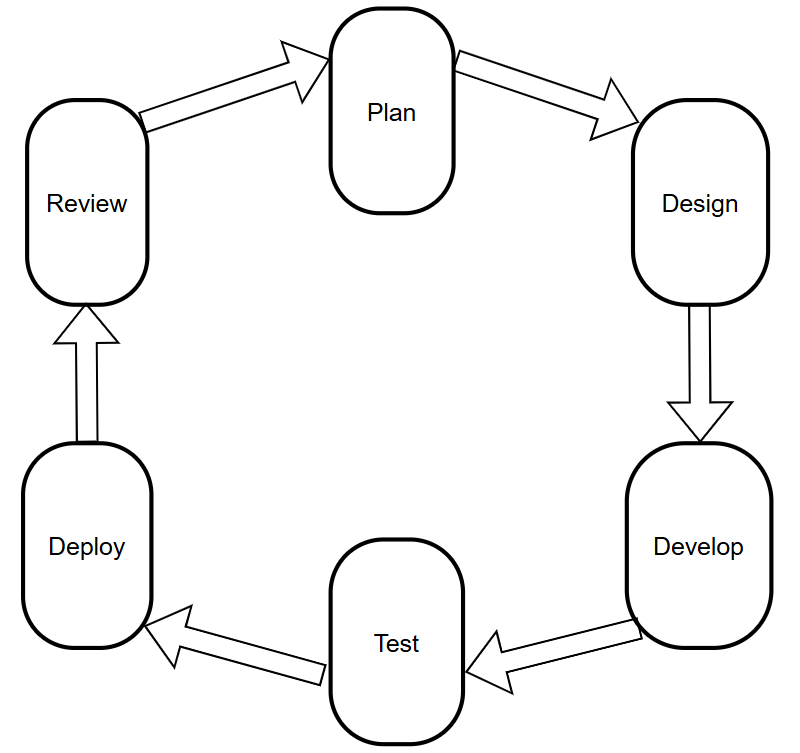
**• Basic Error Recovery:**

While basic validation and error feedback are implemented, advanced rollback, retry logic, and conflict resolution are limited.

## 

## 1.5. Development Methodology

The development of the DevOps Butler platform follows the Agile Software Development Methodology, emphasizing iterative development, continuous feedback, and adaptability to evolving requirements. Agile suits this project well as it enables rapid prototyping and incremental feature enhancements based on real-world testing and user feedback.



**Figure 1.1: Agile Methodology**

DevOps Butler integrates multiple components—responsive frontend UI, FastAPI backend, AI-driven configuration generation, Docker-based deployment orchestration, and secure environment variable handling. These interconnected parts demand ongoing testing and validation to ensure reliable, seamless deployments. Agile’s sprint-based cycles allow the team to develop, test, and refine each feature in manageable increments, facilitating faster delivery and improved overall quality.

## 1.6. Report Organization

This report covers all information related to the DevOps Butler deployment automation platform. The report is organized as follows:

**Chapter 1:** Introduces DevOps Butler, outlining its purpose, core technologies, and the challenges it addresses. This chapter also presents the problem statement, project objectives, and scope.

**Chapter 2:** Provides an analysis of existing deployment methods and tools, highlighting gaps that DevOps Butler aims to fill.

**Chapter 3:** Discusses system analysis and design, including feasibility studies. It covers data and process modeling, along with detailed system design through architecture diagrams, workflows, and interface layouts.

**Chapter 4:** Describes the development tools, frameworks, and technologies used on both frontend and backend. It also explains the project modules, AI integration, and testing procedures including unit and system testing.

**Chapter 5:** Concludes the report with summary, lessons learned during development, and recommendations for future enhancements.

# CHAPTER 2: BACKGROUND STUDY AND LITERATURE REVIEW

## 2.1. Background Study

The increasing complexity of deploying modern full-stack applications has made automated deployment tools essential in the software development lifecycle. DevOps Butler is a local deployment automation platform designed to simplify this process by enabling users to deploy applications from any Git repository with minimal manual setup. It integrates containerization technologies like Docker and Docker Compose with AI-powered analysis to intelligently generate and patch deployment configurations. This combination reduces the need for deep expertise in container orchestration and networking.

DevOps Butler features a modern web interface for submitting repositories, securely handling environment variables, and displaying real-time deployment logs through WebSockets. Behind the scenes, it uses Nginx as a reverse proxy to provide clean, user-friendly local URLs for deployed services. The platform’s AI component helps fix common issues and automates the creation of Dockerfiles and Compose files, making deployment accessible to developers, testers, and teams without complex configuration.

Inspired by existing deployment tools and the need for streamlined local development environments, DevOps Butler balances ease of use with powerful automation. It is built with extensibility in mind, supporting future integrations such as Kubernetes and enhanced monitoring. By automating the most challenging parts of containerized app deployment, DevOps Butler aims to accelerate development workflows and simplify onboarding for teams of all sizes.

## 2.2. Literature Review

The complexity of deploying modern applications efficiently has driven significant research in automated deployment and DevOps tooling. DevOps Butler builds upon this foundation by integrating containerization, automation, and AI-driven configuration to simplify local deployment workflows.

Early deployment automation tools focused on scripting and manual configuration management, using tools like Ansible and Chef to provision environments [1][2]. While effective for server provisioning, these tools required significant expertise and lacked seamless container integration. The rise of Docker introduced a new paradigm for packaging applications, leading to widespread adoption of container orchestration platforms such as Kubernetes [3]. However, Kubernetes’ complexity and steep learning curve created barriers for smaller teams and individual developers [4].

Docker Compose emerged as a simpler alternative to manage multi-container applications, but writing and maintaining Compose files still required understanding of YAML syntax and networking concepts [5]. Recent research emphasizes the need for intelligent automation to generate and validate these configurations automatically. Studies such as “AI-Driven Configuration Generation for Containerized Deployments” demonstrate how AI models can assist in detecting misconfigurations and suggesting fixes, reducing human error and accelerating deployment [6].

DevOps Butler combines these advances by using AI-powered analysis to automatically generate and patch Docker and Docker Compose files from arbitrary Git repositories, reducing manual setup. It also employs WebSocket-based real-time logging and reverse proxy integration, techniques aligned with best practices in modern DevOps tooling [7][8].

By leveraging lightweight frameworks like Flask for backend APIs and integrating container runtimes locally, DevOps Butler offers an accessible platform for developers to deploy, monitor, and manage applications without deep infrastructure knowledge. This reflects ongoing trends toward democratizing DevOps processes and improving developer productivity [9].

# CHAPTER 3: SYSTEM ANALYSIS AND DESIGN

## 3.1. System Analysis

The system consists of several components, each playing a crucial role in the overall functionality. In this section, we analyze the system in terms of functional and non-functional requirements.

### 3.1.1. Requirement Analysis

**i. Functional Requirements:**

Functional requirements define the essential features and behaviors that the **DevOps Butler** system must provide to meet user needs and ensure smooth deployment and management of applications. These requirements focus on automation, user interaction, and real-time monitoring.

**a) Repository Analysis and Dockerfile Detection**

* The system must scan user-provided Git repositories to identify Dockerfiles and Docker Compose files.
* It should analyze the repository structure to detect relevant deployment configurations automatically.

**b) Automatic Dockerfile and Compose Generation**

* When missing, the system must generate Dockerfiles and Docker Compose files using AI-assisted templates based on the project type.
* Generated files should be customizable by users before deployment.

**c) Deployment Management**

* Users must be able to deploy projects locally using Docker or Docker Compose with a single command from the UI.
* The system should manage port assignments and container networking automatically to avoid conflicts.

**d) Real-Time Logs and Monitoring**

* The backend must stream real-time container logs to the frontend via WebSocket for live monitoring.
* Users should be able to view, filter, and download logs during and after deployment.

**e) Reverse Proxy Integration**

* DevOps Butler must automatically configure and update an Nginx reverse proxy to provide clean URLs for deployed services.
* It should handle conflicts and update proxy settings dynamically when new services are added or removed.

**f) User Authentication and Project Management**

* The system should support user login and signup to enable project history and deployment management.
* Users must be able to save, edit, and delete deployment configurations associated with their accounts.

**g) Error Handling and Notifications**

* The system should display clear error messages for failed deployments, build errors, or misconfigurations.
* Users must be notified if container startup fails or resources are unavailable.

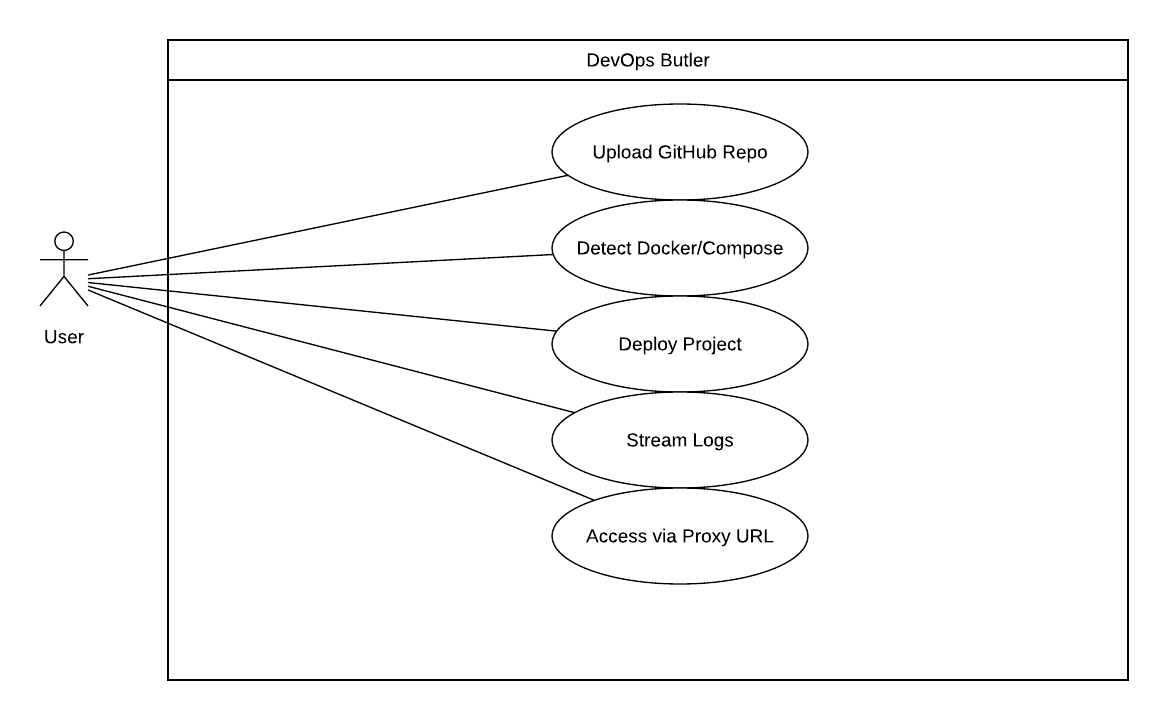
**h) Multi-Project Support**

* Users should be able to deploy multiple projects concurrently without interference.
* The system should track each project’s status independently.

**i) Responsive and Minimal UI**

* The frontend must be responsive and functional across devices and screen sizes.
* Visual indicators such as loading spinners or progress bars should be shown during build and deployment processes.

**• Use Case Diagram**



**Figure 3.1: Use Case Diagram**

Users start by submitting a GitHub repository URL containing a project they wish to deploy. DevOps Butler then scans the repository to detect configuration files—specifically looking for Dockerfile or docker-compose.yml. If found, users are given the option to choose between Docker or Docker Compose as their preferred deployment method. Once the deployment mode is selected, DevOps Butler builds and deploys the application using containerized infrastructure on the host machine. During this process, real-time logs are streamed to the frontend via WebSockets, allowing users to monitor build progress, errors, and service status. Once the deployment completes, DevOps Butler assigns a clean, conflict-free local URL using Nginx reverse proxy, enabling users to access their deployed application directly via browser. If an error occurs at any stage—such as invalid repo format, missing config files, or build failure—an error message is displayed along with diagnostic details. Guest users can deploy a limited number of repositories or services without authentication. When their limit is reached, they are prompted to sign up or log in for extended access. Registered users can deploy multiple projects, access a history of previous deployments, monitor status dashboards (e.g., Grafana if enabled), and manage active services. Login and signup features unlock advanced functionality and enable future integrations such as resource usage tracking, deployment analytics, and project tagging.

#### ii. Non-functional requirement:

* Usability: The system should provide a clean, intuitive web interface for deploying repositories with minimal steps.
* Performance: Docker or Compose-based deployments should initialize within 15–30 seconds under normal conditions.
* Security: API inputs and repository links must be sanitized; user authentication should follow industry security practices.
* Maintainability: The codebase should be modular, with clear separation of concerns to support rapid iteration and scalability.

**3.1.2. Feasibility Analysis**

**i. Technical Feasibility**

The development of DevOps Butler is technically feasible with current DevOps toolchains and open-source technologies. The backend is built using Python (Flask or FastAPI), responsible for scanning repositories, detecting Docker-related files, managing deployment logic, and streaming logs via WebSockets. The application supports Docker and Docker Compose for deploying user projects and uses Nginx as a reverse proxy to provide local access URLs. Frontend components are built using Next.js or React, offering responsive UI for seamless deployment interaction.

The modular architecture supports integration with tools like Grafana for real-time monitoring and future Kubernetes-based deployment using Helm charts. All systems are containerized, ensuring consistency across environments. This makes DevOps Butler highly adaptable, easy to scale, and technically sound for full-stack DevOps automation.

**ii. Economic Feasibility**

DevOps Butler is economically feasible for individual developers, students, or small teams. Core infrastructure relies on free-tier services like GitHub (for repo hosting), Docker Engine (for local containerization), and open-source Nginx. Local development uses minimal system resources and avoids cloud costs. For testing and internal deployment, platforms like Railway, Render, or even self-hosted VPSs can be utilized without major expense.

Logging and monitoring solutions like Grafana and Loki can be self-hosted or integrated using community editions. Optional domain registration and SSL provisioning are low-cost, only necessary for production-grade deployments. The platform’s lightweight architecture and minimal dependency model ensure that operational costs remain low and predictable, aligning well with student or early-stage project budgets.

**iii. Operational Feasibility**

DevOps Butler is operationally feasible and effectively supports the core use case of automated application deployment from GitHub repositories. The system enables users to input a repo URL, choose a deployment method (Docker or Compose), and receive a functioning, locally-accessible service within seconds. It also supports real-time logging for user feedback during builds.

The use of Nginx for reverse proxy, Docker for containerization, and simple UI/UX ensures operational simplicity. No specialized hardware is required, and the system runs smoothly on a typical developer machine or VPS. The platform’s modularity makes it easy to extend—adding support for Kubernetes, credit-limited usage, or team-based access—without disrupting core functionality. Overall, DevOps Butler is operationally sound for educational, testing, and early production usage.

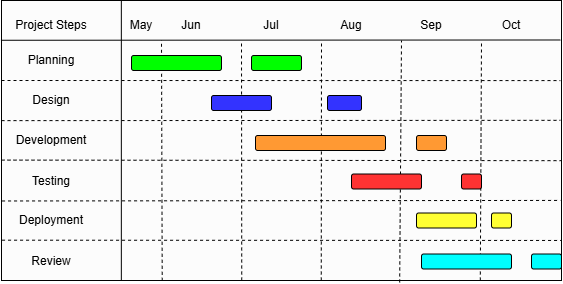
#### iv. Schedule

Schedule time evaluation is the most important consideration in the development of

project. The time schedule requires for the developed of this project is very important.

Proposed system would be designed as per the time calculated

The development of the **DevOps Butler** followed an **Agile methodology**, structured in sprints to allow incremental development, testing, and improvement. The following schedule outlines the key activities and milestones completed during the mid-term phase:

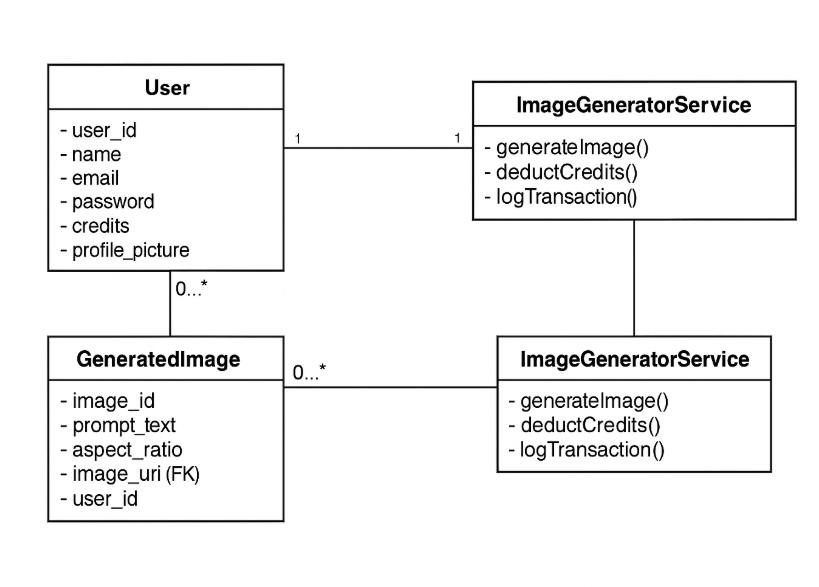


**Table 3.1: Gantt Chart**

### 3.1.3. Object Modelling using Class and Object Diagrams

Object modeling is essential for visualizing the internal structure and relationships among components in the DevOps Butler deployment system. The Class Diagram outlines the static architecture by highlighting core classes such as User, Repository, Deployment, DockerManager, and LogStreamer. Each class includes relevant attributes and methods—such as Repository.url, Deployment.status, and DockerManager.deployCompose()—to reflect the behavior and data managed by the system. Relationships are mapped clearly, for example, showing how a User may own multiple Deployments, or how Deployment interacts with DockerManager to initiate builds.

The Object Diagram supplements this by illustrating real-time instances, such as a specific User deploying a GitHub repo via Compose, with a Deployment object tracking logs and status. These diagrams help clarify the architecture, assist in aligning with functional goals, and guide development by offering a blueprint of system behavior and entity interaction.



**Figure 3.3: Class Diagram**

**3.1.4. Dynamic Modelling using State and Sequence Diagrams**

Dynamic modeling captures the operational flow and behavioral changes of the DevOps Butler system over time. The State Diagram reflects the various states of a repository lifecycle—from being detected, selected, analyzed, containerized, deployed, to monitored—while also covering transitions like error handling or rollback. This helps illustrate how DevOps Butler handles different repository types, deployment preferences (Docker/K8s), and failure states. The Sequence Diagram models the interactions between the user interface, backend engine, detection service, deployment engine, and monitoring dashboard. It showcases how a user's input (like GitHub URL) initiates the sequence: file analysis, Dockerfile/Compose detection, containerization, deployment, logging, and result visualization. These diagrams are essential for understanding the execution flow, service orchestration, and identifying integration points or delays.

**3.1.5. Dynamic Modelling using Activity Diagram**

Activity diagrams visually represent the workflow of DevOps Butler, demonstrating how user actions and backend logic are coordinated to perform intelligent deployments. The diagram begins with the user entering a GitHub repository link. DevOps Butler then scans for Docker-related files and determines whether to deploy using Docker Compose or Helm charts for Kubernetes. Depending on the result, the system builds the container image or applies manifests. The system also routes traffic via Nginx and displays real-time logs. If no deployment files exist, the AI assistant generates them automatically. Users are presented with deployment status, exposed URLs, and resource usage. Key decision points include file detection, deployment method selection, and monitoring preferences. This activity flow emphasizes user interaction, automation logic, and error handling, providing a clear view of system behavior from input to deployment and monitoring.

## 3.2. System Design

A good system design is to organize the program modules in such a way that are easy to develop and change.

**3.2.1 Refinement of Class, Object, State, Sequence, and Activity Diagrams**

As DevOps Butler progressed from initial design to implementation, iterative refinement of key UML diagrams was essential to accurately model the system’s architecture and behavior. The class diagram was updated to include critical attributes such as deployment metadata, user upload sessions, container states, and AI-generated config details. Relationships between classes like DeploymentRequest, Workspace, Container, NginxConfig, and AIAnalyst were optimized to clearly define responsibilities and data flow. The object diagram was refined to demonstrate real-time runtime instances — for example, a user deploying a MERN stack app with associated containers and Nginx proxy configuration. The state diagram was improved to model deployment lifecycle states, including Initialized → Cloned → Analyzed → Built → Deployed → Running → Destroyed, with transitions triggered by actions like starting deployment, AI patching, container startup, and cleanup.

The sequence diagram detailed interactions between the Web Frontend, FastAPI backend, AI Analyst module, Docker engine, and Nginx proxy, illustrating request handling, AI-based config generation, container orchestration, and log streaming step-by-step.

Finally, the activity diagram highlighted key conditional workflows such as environment variable injection, port conflict resolution, AI generation fallback, and cleanup routines. These refinements ensured consistency between design and implementation, improving developer communication and system maintainability.

**3.2.2 Component Diagrams**

Component diagrams visualize the modular organization and dependencies within DevOps Butler. The system is divided into loosely coupled, interacting components:

* Web Frontend: Handles user inputs such as Git repo URLs and .env uploads, and displays deployment status/logs in real-time.
* FastAPI Backend: Acts as the orchestration core, managing deployment requests, workspace lifecycle, AI integration, Docker commands, and Nginx config generation.
* AI Analyst Module: Provides intelligent Dockerfile and Compose file generation or patching using AI models.
* Docker Engine: Executes container build and runtime commands.
* Nginx Proxy (Dockerized): Routes HTTP traffic to deployed services using dynamically generated configs.
* SQLite Database: Stores deployment metadata, history, and status without storing secrets.

Each component communicates through clearly defined APIs or Docker network interfaces, enhancing scalability and maintainability. This modular design facilitates easy extension, such as adding support for Kubernetes or new service integrations in the future.

**3.2.3 Deployment Diagrams**

Deployment diagrams describe the physical distribution of DevOps Butler’s components across infrastructure nodes:

* The Web Frontend runs in the user’s browser, served from a local or remote static file server.
* The FastAPI Backend is hosted on a local machine or server (e.g., a Mac or Linux box), orchestrating deployments.
* The Docker daemon runs locally or remotely, handling container lifecycle commands issued by the backend.
* The Nginx Proxy is deployed as a Docker container on the same host, dynamically reconfigured by the backend to route traffic to running containers.
* The AI Analyst uses API access to external AI services (e.g., OpenAI/Gemini) for config generation, either locally or via cloud endpoints.
* The SQLite database is embedded within the backend environment to store deployment metadata securely.

This distributed deployment model optimizes for local developer environments while maintaining clear separation of concerns and scalability paths. Secure network communication, container isolation, and ephemeral workspaces ensure robustness and security throughout the deployment lifecycle.

# CHAPTER 4: IMPLEMENTATION AND TESTING

## 4.1. Implementation

This section talks about the implementation of the system.

### 4.1.1. Tools Used

**i. CASE Tools**

* **Cursor**

Cursor has been used extensively for writing, revising, and refining the project’s codebase and documentation. Its AI-powered assistance helps improve code quality and accelerates development by providing intelligent suggestions and error detection.

* **Lucidchart**

Lucidchart was used to design, update, and finalize all architecture, workflow, and UML diagrams for DevOps Butler. Its collaborative features and rich diagramming tools ensured clear visualization of system components and processes.

* **Microsoft Word**

Microsoft Word served as the primary tool for creating and formatting the project documentation. Its robust word processing capabilities helped produce well-structured, professional, and easily maintainable documentation for DevOps Butler.

**ii. Programming Languages**

* **Python**

Used to develop the FastAPI backend server, orchestrate the deployment pipeline, handle API requests, and manage interactions with Docker, Nginx, and AI services for config generation.

* **HTML, CSS, JavaScript**

Used to build the responsive web frontend that allows users to submit Git repositories, upload .env files, and view real-time deployment status and logs.

**iii. Frameworks & Libraries:**

* **FastAPI**

**A high-performance Python web framework used to build the backend API that handles deployment requests, orchestrates the pipeline, and manages real-time communication with the frontend.**

* **Docker SDK / Docker Compose**

**Used to programmatically manage container lifecycle, build images, run containers, and orchestrate multi-container deployments.**

* **Tailwind CSS / Custom CSS**

**Used for creating a modern, responsive, and user-friendly frontend design.**

**iv. Model & Backend Infrastructure:**

* **Local or Cloud Host (e.g., Mac, Linux Server)**

Runs the FastAPI backend, Docker daemon, and Nginx proxy to manage containerized deployments and traffic routing.

* **AI Services (OpenAI / Gemini API)**

Provides AI-powered analysis and generation of Dockerfiles and Docker Compose configurations to automate and optimize deployment setups.

**v. Database Platforms**

1. **SQLite**  
   A lightweight, file-based relational database used to store user information, image metadata, and credit history.

### 4.1.2. Algorithms

The core algorithm behind DevOps Butler automates the local deployment of full-stack applications by intelligently managing containerized environments using Docker, Docker Compose, Nginx, and AI-assisted configuration generation.

**1. Deployment Pipeline**

DevOps Butler orchestrates the deployment workflow starting from a Git repository URL and optional environment files (.env). It automates cloning, analysis, container orchestration, and traffic routing to provide a seamless deployment experience accessible via pretty local URLs.

* *Validation & Workspace Setup*

Validates the Git URL, creates an isolated temporary workspace to avoid conflicts between deployments.

* *Repository Cloning & Analysis*

Clones the repository, scans for Dockerfiles, docker-compose files, and environment variables. Warns if .env files exist in the repo, preferring user-uploaded ones.

* *AI-Powered Configuration Generation & Patching*

If Docker artifacts are missing or need fixes, AI is invoked to generate or patch Dockerfiles and Compose files, addressing best practices, networking, and port conflicts.

* *Dynamic Port Assignment & Environment Injection*

Assigns free host ports dynamically to avoid collisions and patches Compose files to use the provided environment variable files securely.

* *Container Orchestration*

Runs docker compose up --build -d to build and start containers on the dedicated Docker network for proper proxying.

* *Nginx Reverse Proxy Configuration*

Generates and reloads Nginx configs to route incoming requests to the correct container with clean URLs.

* *Real-Time Status & Log Streaming*

Streams logs and deployment status to the frontend using WebSockets for live feedback.

* *Cleanup & Lifecycle Management*

On destroy or redeploy, removes containers, deletes configs, and cleans temp files to maintain a tidy environment.

**2. AI-Assisted Compose Patching**

The AI receives the original Compose file and outputs a patched version that:

* Ensures use of an external Docker network (devops-butler-net)
* Fixes port conflicts by assigning available ports
* Adds references to environment files securely
* Applies container orchestration best practices
* The patched file replaces the original for deployment.

**3. Dynamic Port Assignment**

When a desired host port is unavailable, the algorithm attempts to bind to incrementing ports until it finds a free one, preventing runtime conflicts.

**4. Secure Environment Variable Injection**

DevOps Butler injects user-uploaded environment files into container configurations, never storing secrets in databases or version control, ensuring safe secret management.

**5. Nginx Configuration Generation & Reload**

For each service, an Nginx config proxies traffic to the actual running container and port. The Nginx proxy container is reloaded to apply new configurations without downtime.

**6. Cleanup Algorithm**

Stops and removes containers, deletes Nginx configs, and removes workspace files, ensuring no residual artifacts remain after deployment termination.

**7. Security Enforcement**

The system warns users about .env files found in repositories and enforces usage of uploaded .env files. Secrets are handled carefully to prevent leakage in logs or persistent storage.

### 4.1.3. Implementation of modules

The DevOps Butler deployment platform is divided into several modular components, each handling specific responsibilities to ensure maintainability, scalability, and ease of development.

**1. User Interface Module**

* **Description:** Manages all user interactions for deployment input, status monitoring, and log visualization.
* **Implementation:**
  + Built using Cursor for the frontend UI and Lucid for diagram visualizations.
  + Provides fields for Git repo URL, environment file uploads, and deployment controls.
  + Displays real-time deployment logs and status via WebSockets.
  + Allows users to manage active deployments and view detailed container info.

**2. Authentication Module**

* **Description:** Handles user authentication and session management.
* **Implementation:**
  + Uses Microsoft Word for project documentation; authentication logic handled backend (e.g., Flask or FastAPI).
  + Supports user login, registration, and session tracking (token or cookie-based).
  + Provides role-based access for deployment permissions and history viewing.

**3. Deployment Pipeline Module**

* **Description:** Automates the cloning, analyzing, building, and deploying of applications from Git repositories.
* **Implementation:**
  + Validates Git URLs and creates isolated temporary workspaces for each deployment.
  + Clones repos and inspects project structure for Dockerfiles and Compose files.
  + Invokes AI-powered logic to generate or patch missing/incorrect Docker Compose files.
  + Handles dynamic port assignment and environment variable injection.
  + Runs container orchestration commands and manages lifecycle events.

**4. Configuration & AI Integration Module**

* **Description:** Interfaces with AI services to patch or generate Docker Compose files and ensure best practices.
* **Implementation:**
  + Sends Compose files to AI with instructions for networking, port conflict resolution, and env file inclusion.
  + Receives, sanitizes, and applies AI-generated configuration patches before deployment.

**5. Reverse Proxy & Networking Module**

* **Description:** Generates and manages Nginx proxy configurations for routing traffic to containers.
* **Implementation:**
  + Creates per-service Nginx config files pointing to container names and internal ports.
  + Reloads the Nginx proxy container dynamically to apply new configurations without downtime.

**6. Real-Time Logging & Status Module**

* **Description:** Streams deployment logs and status updates to the frontend in real-time.
* **Implementation:**
  + Uses WebSockets to deliver live logs and deployment state feedback.
  + Enables users to monitor builds, container startups, and network configuration in a unified interface.

**7. Cleanup & Resource Management Module**

* **Description:** Manages cleanup of containers, temporary files, and Nginx configs upon deployment termination.
* **Implementation:**
  + Stops and removes containers associated with deployments.
  + Deletes generated Nginx proxy files and temporary workspace directories.
  + Updates deployment status in persistent storage or logs.

**8. Security & Environment Handling Module**

* **Description:** Ensures secure handling of environment variables and sensitive files.
* **Implementation:**
  + Warns users if .env files are detected in repos and prioritizes uploaded .env files.
  + Prevents storage of secrets in databases or logs.
  + Injects environment variables into Docker Compose securely during deployment.

## 4.2. Testing

This project has been tested thoroughly to ensure proper working.

**Test Case 1: Successful Deployment from GitHub Repository**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S.N | Inputs | Expected Result | Actual Results | Pass/Fail |
| 1 | GitHub Repo URL: <https://github.com/user/sample-mern-app>  .env files: frontend.env & backend.env | Application is cloned, containers are started, and accessible at http://sample-mern-app.localhost:PORT | As Expected | Pass |

**Table 4.1: Successful Deployment from GitHub**

This test confirms that providing a valid Git repository and required environment files results in successful containerized deployment with proper proxy configuration.

**4.2.2. System Testing**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| S.N | Test Scenario | Inputs | Expected Result | Actual Results | Pass/Fail |
| 1 | Full deployment lifecycle validation | Git Repo URL: <https://github.com/user/sample-mern-app>  .env files: frontend.env & backend.env | Application deployed, accessible locally with correct ports and Nginx routing | As Expected | Pass |

**Table 4.2: System Testing**

The system was tested end-to-end — from repository validation and AI patching to Docker orchestration and Nginx proxying. The deployment succeeded and functioned as expected, demonstrating DevOps Butler’s ability to handle real-world full-stack application deployments smoothly.

# CHAPTER 5: CONCLUSION AND FUTURE RECOMMENDATIONS

The DevOps Butler platform has successfully established a robust and intelligent system for local containerized application deployment using Docker, Docker Compose, and AI assistance. By integrating a FastAPI backend with real-time WebSocket logging, an AI-driven configuration engine, and a responsive web frontend, the project simplifies the complexity of deploying full-stack applications from Git repositories. Core features such as environment variable injection, AI-powered Dockerfile/Compose generation, dynamic port management, and Nginx-based reverse proxy routing have been implemented and validated.

While future enhancements like Kubernetes support, Helm-based deployments, Grafana integration, and multi-user access control are in the pipeline, the current phase demonstrates the system’s core value and feasibility. This milestone proves DevOps Butler's potential as a developer-first tool that automates the hardest parts of local dev environments.

The development and documentation process sharpened our skills in backend orchestration, DevOps practices, containerization, and AI-assisted automation—forming a strong foundation for the final product.

## 5.2. Future Recommendations

As DevOps Butler evolves toward a more mature and extensible platform, the following enhancements are recommended to improve automation, flexibility, and user experience:

* **Kubernetes Support**

Integrate Kubernetes for deploying microservices at scale, offering more control over orchestration, scaling, and monitoring.

* **Helm Chart Integration**

Enable Helm-based deployments for advanced users to customize and deploy Kubernetes resources more efficiently.

* **Grafana Monitoring Dashboard**

Embed Grafana into the frontend to visualize deployment metrics, container health, and resource usage in real time.

* **Multi-User Access & Roles**

Introduce authentication with role-based access control to allow teams to collaborate and manage deployments securely.

* **Project Rollback & Versioning**

Add the ability to rollback failed deployments and maintain a history of changes for auditing and recovery.

* **Third-Party Service Integration**

Support integration with services like PostgreSQL, Redis, and message brokers through optional Compose extensions.

* **User Feedback & Error Reporting**

Implement a feedback and error reporting system to gather insights and improve platform stability continuously.

# REFERENCES

[1] J. Cohen et al., “Infrastructure as Code: Managing Servers with Configuration Management,” O’Reilly, 2017.

[2] A. Hasselgren, “Ansible for DevOps,” 2nd ed., 2020.

[3] B. Burns et al., “Kubernetes: Up and Running,” O’Reilly, 2018.

[4] M. Hightower et al., “Kubernetes Best Practices,” O’Reilly, 2019.

[5] S. Turnbull, “The Docker Book,” 2018.

[6] R. Smith et al., “AI-Driven Configuration Generation for Containerized Deployments,” IEEE Transactions on Cloud Computing, 2022.

[7] D. Merkel, “Docker: Lightweight Linux Containers for Consistent Development and Deployment,” Linux Journal, 2014.

[8] P. Hunt, “WebSockets and Real-Time Applications,” Packt Publishing, 2021.

[9] N. Leung, “DevOps Automation: Principles and Practices,” Apress, 2020.