
# Tribhuvan University

## Faculty of Humanities and Social Sciences

# A PROJECT REPORT ON

# "DevOps Butler"

**Submitted to**

**Department of Computer Application**

**Reliance College**

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

Submitted by

Aayush Adhikari

Reg No:- 6-2-522-172-2020

August, 2025

Under the Supervision of

**Abhijeet Kumar Sah**


# Tribhuvan University

## Faculty of Humanities and Social Sciences

## 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 44600

# Tribhuvan University

# Faculty of Humanities and Social Sciences

# Reliance College

## LETTER OF APPROVAL

This is to certify that this project prepared by Sanskar Satyal 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.

| | |
|---|---|
| ……<br>**Abhijeet Kumar Sah**<br>**Supervisor**<br>Chabahil, Saraswatinagar, Kathmandu 44600 | ……<br>**Mr. Ayush Dhital**<br>**Coordinator**<br>Chabahil, Saraswatinagar, Kathmandu 44600 |
| ……<br>**Internal Examiner** | ……<br>**External Examiner** |

# ABSTRACT

To be successful, modern deployment platforms must feature a simplified user experience, automated build processes, robust container management, real-time feedback, and complete lifecycle control. Deployment systems can range from complex, enterprise-level CI/CD pipelines to simpler, managed Platform-as-a-Service (PaaS) offerings, each with varying levels of configuration overhead. DevOps Butler aims to introduce a new, self-hosted deployment assistant that transforms Git repositories into running containerized applications with zero configuration. The platform provides a modern, intuitive web interface, allowing developers to initiate deployments with a single click by providing a Git repository URL. DevOps Butler will intelligently analyze the repository, automate the Docker build and deployment process, and stream live, color-coded logs via WebSockets to ensure a transparent and seamless developer experience. Overall, DevOps Butler aims to deliver a convenient and powerful local PaaS experience, abstracting away the complexities of container orchestration while establishing a robust and extensible deployment framework.

***Keywords: Automation, Deployment, Containerization, Docker, PaaS, CI/CD, FastAPI***

# ACKNOWLEDGEMENT

I would like to express our sincere gratitude to all those who have contributed to this project. First and foremost, we are grateful to our project supervisor, **Abhijeet Kumar Sah,** who provided us with invaluable guidance, feedback, and support throughout the entire project. His knowledge, expertise, and dedication were instrumental in helping us to develop and execute our research project successfully.

I would also like to extend our thanks to **Abhijeet Kumar Sah** who served as members of our project committee. Their insightful feedback, constructive criticism, and suggestions were crucial in helping us to refine our project goals, research methodology, and analysis.

A special note of gratitude goes to the participants of my study. Their willingness to take time out of their schedules to share their experiences, perspectives, and insights made this project possible. I truly appreciate their openness and contribution, and I feel honored to have had the opportunity to work with them.

Additionally, I am grateful to my colleagues for their feedback and suggestions throughout the course of the project. Their encouragement, support, and constructive input helped keep me motivated and focused from start to finish.

Sincerely,
Aayush Adhikari
DevOps Butler

# Table of Contents

# List of Figures

# List of Tables

# List of Abbreviations

HTML : Hypertext Markup Language

CSS : Cascading Style Sheets

JS : JavaScript

CRUD : Create, Read, Update, Delete

MVC : Model-View-Controller

UTA : User Acceptance Testing

CASE : Computer Aided Software Engineering

DFD : Data Flow Diagram

ER : Entity Relationship

UI : User Interface

UX : User Experience

# Chapter 1: Introduction

## 1.1. Introduction

The The landscape of software development and deployment has shifted dramatically, moving towards automated, container-based workflows. Today's developers and operations teams require tools that eliminate manual configuration, streamline the path from source code to a running application, and provide immediate, transparent feedback. In response to this, "DevOps Butler" is conceived as a sophisticated, automated deployment assistant designed to provide a holistic ecosystem for containerizing and managing applications with zero initial setup. As the reliance on container technologies like Docker grows, the necessity for a user-centric, automated deployment platform is no longer just an advantage but a fundamental requirement for efficient and reliable software delivery.

This project documents the architecture and implementation of a comprehensive deployment solution designed to simplify this complex domain. The platform is built upon the foundational principles of modern DevOps: automation, idempotency, and observability. DevOps Butler internalizes these principles, aiming not only to execute deployment tasks but to foster a seamless and powerful developer experience by abstracting away the underlying complexities of container lifecycle management, networking, and monitoring.

The essence of this deployment portal revolves around the automation and orchestration of the entire deployment pipeline. Furthermore, the project endeavors to enhance the developer experience by providing real-time, live-streamed logs and a modern web interface for managing applications. Collectively, the project seeks to elevate the deployment process through intelligent automation, real-time observability, and a complete container management infrastructure.

The project aims for the following:

- Providing one-click deployments directly from a Git repository URL.
- Automating the analysis of project structure to support Dockerfile and Docker Compose configurations.
- Managing the complete Docker container lifecycle, including building, running, networking, and cleanup.
- Ensuring a transparent and debuggable process through real-time log streaming and deployment history tracking.

## 1.2. Problem Statement

In modern software development, developers face significant challenges in deploying their applications, such as complex configuration, repetitive manual steps, and a general lack of visibility into the build process. These challenges can be addressed by developing a robust automated deployment system that will help developers streamline the path from source code to a running application. The purpose of any automated deployment platform is to help developers abstract away the underlying complexities of containerization and orchestration, enabling them to focus on writing code rather than managing infrastructure.

The primary purpose of DevOps Butler is to assist developers by automating the entire deployment pipeline for containerized applications. This process begins with a simple, user-friendly web interface where a developer can provide a Git repository URL. For example, suppose a developer has an application with a Dockerfile in its root directory. His or her deployment process should be as simple as pasting the URL and clicking a "Deploy" button. As the platform takes over, its intelligent analysis should automatically detect the Dockerfile or docker-compose.yml, choose the correct build strategy, and manage container networking and port mapping. As the system executes these steps, the developer should receive real-time, color-coded logs, providing complete transparency and making it easy to debug any issues.

In conclusion, a well-designed deployment assistant like DevOps Butler is vital for overcoming the challenges of modern application deployment. By creating a developer-centric platform that automates the build, deployment, and container lifecycle management processes, we can effectively reduce configuration errors, increase deployment speed, and enhance overall developer productivity in the ever-growing containerized ecosystem.

## 1.3. Objectives

The main objectives of this project is as follow:

1 To provide a zero-configuration, one-click deployment system that fully automates the process of turning a Git repository into a running containerized application.

2 To manage the entire container lifecycle, including intelligent project analysis, building, networking, port mapping, and a complete destroy functionality for clean resource management.

3 To deliver a superior developer experience through a modern web interface that provides real-time deployment logs, status tracking, and a persistent history of all operations.

## 1.4. Scope and Limitation

**Scope**

Scope

The project is a Python-based deployment assistant for containerizing applications, developed using FastAPI. The system allows users to automate the deployment of Git repositories by providing a URL, while the platform handles the entire containerization pipeline. Here are some key aspects within the scope of DevOps Butler:

1) **Git-Based Deployment:**
    i) The system allows users to deploy applications by providing a public Git repository URL through a modern web interface or a programmatic POST request.
    ii) It clones the repository into a temporary directory to begin the deployment process.
2) **Automated Project Analysis:**
    i) DevOps Butler automatically analyzes the root directory of the cloned repository to detect either a Dockerfile or a docker-compose.yml file to determine the correct deployment strategy.
3) **Container Lifecycle Management:**
    i) The system manages the full container lifecycle, including building Docker images, running containers on a custom network (devops-butler-net), and discovering and mapping ports.
    ii) It provides a "destroy" function to completely and cleanly remove all associated resources, including containers and database records, ensuring idempotency.
4) **Real-Time UI & Logging:**
    i) The system features a web interface for initiating deployments and viewing application status.
    ii) It uses WebSockets to stream live, color-coded build and deployment logs directly to the user's browser, providing real-time feedback.
5) **Deployment History and Status:**

i) A persistent SQLite database is used to track the history of all deployments, including their status (e.g., 'starting', 'success', 'failed'), timestamps, and container details.

6) **API Endpoints:**

i) The platform exposes a RESTful API for programmatic control, allowing users to deploy, list all deployments, and destroy specific applications via HTTP requests.

**Limitation**

On the other hand, limitations specify any constraints or restrictions within the DevOps Butler platform. Some limitations in the project are:

1) Limited Repository Structure Detection: The system can only detect a Dockerfile or docker-compose.yml file if it is located in the absolute root of the repository. Detection of these files within subfolders is not yet implemented.
2) No Advanced Deployment Features: The platform does not currently support multi-environment configurations (e.g., staging, production), custom domains, or automatic SSL/TLS certificate generation.
3) Lack of Application Monitoring: There is no built-in functionality for performing automatic health checks on deployed applications or for monitoring resource usage like CPU, memory, and disk.
4) Basic Project Analysis: The "intelligent analysis" is currently limited to file detection. It does not yet support AI-generated Dockerfiles or automatic dependency detection for projects that lack an existing Docker configuration.

## 1.5. Development Methodology

The development of "DevOps Butler" follows the Agile methodology, which emphasizes iterative progress, continuous feedback, and adaptability. Agile is particularly suited to this project due to its flexibility in building a complex system with interconnected components, allowing for the delivery of a functional platform in manageable increments.

1) **Plan**

In the planning phase, I planned for the system I was willing to make, focusing on the core needs of a developer looking for an automated deployment solution. This involved breaking down high-level requirements into specific user stories. For example, I identified user stories such as “As a developer, I want to deploy an

application simply by pasting its Git URL," and "As a user, I want to view live logs of the deployment process to understand what is happening." This groundwork set a clear direction for the subsequent phases of development.

2) **Design**

During the design phase, I architected the core components of the system, choosing FastAPI for the backend, Vanilla JavaScript for the frontend, WebSockets for real-time communication, and Docker as the containerization engine. I conducted sprint planning sessions to prioritize user stories and allocate tasks. Each sprint was structured to last 1–2 weeks, focusing on specific goals such as building the initial API endpoints for deployment, implementing the repository cloning and analysis logic, or integrating the WebSocket for live logging. This phase allowed me to outline a clear technical roadmap for each iteration.

3) **Develop**

The development phase was characterized by iterative cycles where I focused on completing core functionalities during each sprint. For instance, one sprint concentrated on implementing the basic Dockerfile deployment strategy, while another focused on adding support for docker-compose.yml. Subsequent sprints introduced the persistent database with SQLModel for deployment history and the complete "destroy" functionality for resource cleanup. By developing in small increments, I was able to regularly test and refine each component, ensuring it met the desired functionality and quality standards.

4) **Test**

In the testing phase, I implemented manual and integration tests to verify the end-to-end functionality of the deployment pipeline. Testing involved using a variety of public sample Git repositories (containing both Dockerfile and docker-compose.yml files) to ensure the analysis, build, and port mapping logic worked correctly. Additionally, user acceptance testing (UAT) was performed by interacting with the web interface to confirm that the deployment process was initiated correctly, logs were streamed accurately, and the final application was accessible via its mapped port. This thorough testing helped identify and address bugs related to command execution, error handling, and state management.

5) **Deploy**

The deployment phase for the development of DevOps Butler itself involved running the application locally in a development environment after each sprint. Using tools

like uvicorn with --reload enabled an efficient feedback loop for testing new features in a real-world setting. I closely monitored the application's interaction with the Docker engine and the local filesystem, ensuring that container creation, networking, and cleanup were performed reliably without creating stale resources or conflicts.

6) **Review**

Finally, the review phase emphasized continuous self-assessment and refinement. I gathered insights after each development cycle by using the tool to deploy my own test projects. This feedback directly influenced the project's direction and the prioritization of features listed in the "Future Roadmap," such as the need for enhanced cleanup logic and a persistent database. Regular reviews allowed me to reflect on the architecture, improve the robustness of the error handling, and ensure the final product effectively meets its core objective of simplifying deployments.

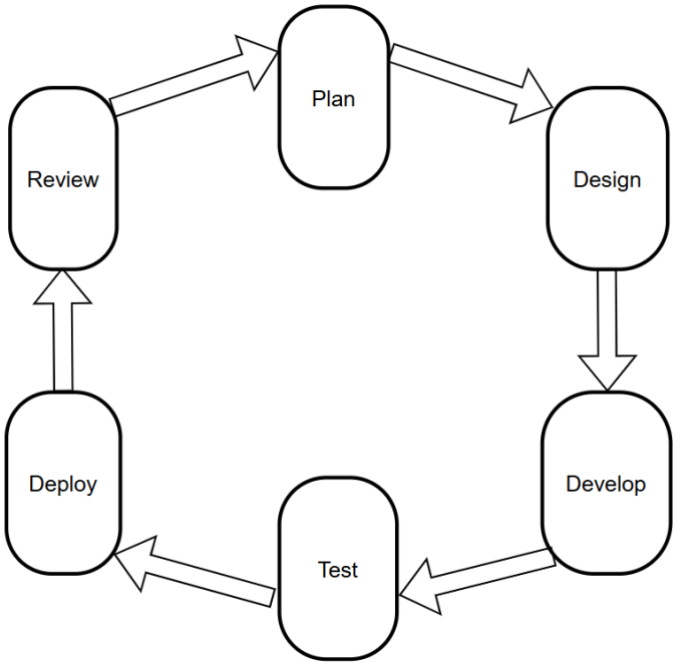



**Figure 1.1: Agile Methodology**

## 1.6. Report Organization

This report is divided into five chapters. Each chapter is further divided into different headings. The preliminary section contains the overall information about the project. This section includes the abstract, table of contents, list of figures, and abbreviations.

1) **Chapter 1** gives an introduction to DevOps Butler. The problem definition, objectives, scope, and limitations of this system are discussed.

2) **Chapter 2** contains a background study of the system and a literature review section where existing technologies and concepts in the fields of automated deployment, Platform-as-a-Service (PaaS), and container orchestration are discussed in brief.
3) **Chapter 3** discusses in detail the analysis and design of the system. It provides information about the existing manual deployment processes, requirements analysis, feasibility study, and required system configuration. It also gives information about the overall system architecture, the deployment pipeline workflow, and the database schema design.
4) **Chapter 4** gives information about the development methodology, implementation techniques using Python and FastAPI, the tools used in the tech stack, and the testing procedures for the system's core functionalities.
5) **Chapter 5** includes the future scope of the project based on the official roadmap and necessary future recommendations along with the conclusion.

# Chapter 2: Background Study and Literature Review

## 2.1. Background Study

A background study is a crucial part of an automated deployment platform's documentation as it provides an in-depth understanding of the software development industry, modern DevOps trends, and the target audience of developers. It involves conducting comprehensive research to gather relevant information that informs the design, development, and operation of the platform. The study begins by analyzing the deployment automation industry, examining the latest trends in containerization, the market size of Platform-as-a-Service (PaaS) offerings, and their growth potential [1].

This research helps identify the competitive landscape and potential opportunities or challenges that a platform like DevOps Butler may face in the market. It ensures that the platform can capitalize on growing trends such as the demand for "zero-configuration" pipelines, improved developer experience (DevEx), and the increasing adoption of Docker for both local development and production environments [2].

Moreover, the background study includes an evaluation of successful deployment and PaaS platforms in the industry [3]. By examining their best practices, user workflows, and innovative features, DevOps Butler can draw inspiration from established services. This analysis allows the platform to incorporate effective strategies, such as one-click Git-based deployments, real-time log streaming, and seamless container lifecycle management, to ensure a competitive edge and high utility for its users.

Conducting a thorough background study ensures that the development team gains valuable insights to make informed decisions throughout the platform's creation process. By aligning the platform with industry trends, understanding the deployment needs of the target developer audience, and integrating proven DevOps methodologies [4], DevOps Butler can deliver a compelling, user-friendly experience that meets developer expectations and thrives in the dynamic world of automated software delivery.

## 2.2. Literature Review

Automated deployment platforms serve as orchestrators for software delivery, managing the complex interactions between source code repositories, build environments, and container runtimes over a network [5]. These platforms are critical in modern software engineering for enabling Continuous Integration and Continuous Deployment (CI/CD) practices. As noted in the literature, the primary goal of such automation is to make

deployments predictable, repeatable, and less error-prone, thereby increasing developer velocity and system reliability [6]. By abstracting away manual steps, these platforms allow developers to focus on core application logic rather than infrastructure management.

Emerging technologies, particularly containerization with tools like Docker, have fundamentally reshaped the deployment landscape. The literature suggests that containerization solves the "it works on my machine" problem by packaging an application with all its dependencies into a single, portable unit [7]. This shift is particularly relevant for deployment platforms like DevOps Butler, as it provides a standardized target for building and running applications, regardless of the underlying programming language or framework.

Furthermore, a well-designed deployment platform must prioritize the developer experience (DevEx). The research asserts that reducing cognitive load through "zero-configuration" or "convention over configuration" paradigms is essential for adoption and efficiency [8]. By automatically analyzing a project's structure and inferring the correct build strategy, a platform can significantly lower the barrier to entry for developers, making the deployment process nearly invisible.

Observability and real-time feedback play a crucial role in the success of automated systems. It is emphasized that providing immediate, transparent feedback—such as live-streaming logs from the build and deployment process—is critical for debugging and building trust in the automation [9]. For deployment platforms, these mechanisms are essential for maintaining a high level of developer confidence and ensuring that failures can be diagnosed quickly.

Resource management and isolation are paramount for system stability. The importance of implementing robust lifecycle controls to create, manage, and cleanly destroy all deployment-related artifacts is widely discussed [10]. Building trust is achieved by ensuring idempotent operations, where deploying the same application multiple times does not lead to resource conflicts or orphaned containers, thus guaranteeing a clean state.

Continuous innovation is vital for platforms in the DevOps space. As development practices and technologies evolve, platforms must adapt by offering new features, such as support for more complex architectures, enhanced monitoring, and integration with cloud providers [11]. This commitment to ongoing development ensures that the platform remains competitive and relevant in the rapidly changing landscape of software engineering.

Feedback mechanisms are integral to the design of an effective deployment platform. The interactive cycle between a developer and the system remains incomplete until the platform promptly responds to a user's action, such as clicking "Deploy." As noted in human-computer interaction studies, feedback that provides users with information about their actions and the system's status is essential [12]. Without it, users may feel uncertain about the process, undermining the overall goal of seamless automation.

In summary, the literature emphasizes the multifaceted nature of automated deployment platforms, where effective orchestration, modern containerization technologies, a superior developer experience, robust observability, clean resource management, and immediate feedback mechanisms are crucial for creating a successful and reliable software delivery system.

# Chapter 3: System Analysis and Design

## 3.1. System Analysis

### 3.1.1. Requirement Analysis

The requirements are to be collected before starting projects development life cycle. To design and develop system, functional as well as non-functional requirement of the system has been studied.

### i. Functional Requirements

The functional requirements for DevOps Butler define the core operations of the platform, such as user input, the automated pipeline processing, and the final output of a running application. The functional requirements of DevOps Butler are mentioned below:

a) **Deployment Functionality:**

- Deploy from Git URL: Users must be able to initiate a new deployment by providing a public Git repository URL through the web interface.
- Automatic Project Analysis: The system must automatically analyze the cloned repository to detect a Dockerfile or docker-compose.yml file in the root directory to determine the deployment strategy.
- Automated Build & Deploy: The system must execute the necessary Docker or Docker Compose commands to build the container image and start the application.
- Real-time Log Streaming: Users must be able to view live, color-coded logs of the entire build and deployment process streamed via a WebSocket connection.
- Port Management: The system must automatically detect exposed container ports and map them to the host, providing an accessible URL for the deployed application.

a) **Lifecycle Management & Monitoring Functionality:**

- View Deployed Applications: Users must be able to view a list or dashboard of all currently running applications/containers via the web interface.
- View Deployment History: The system must maintain and display a persistent history of all deployments, including their status (e.g., success, failed) and timestamps.

- Destroy Deployment: Users must have the ability to completely destroy a deployment, which stops and removes the associated containers and cleans up all related database records.
- Idempotent Deployment Handling: The system must automatically clean up any previous, existing deployment of the same repository before starting a new one to prevent resource conflicts.

**a) API Functionality:**

- Programmatic Deployment: The system must expose a POST endpoint (/deploy) to allow for initiating deployments via API calls.
- List Deployments: The system must provide a GET endpoint (/deployments) to allow programmatic retrieval of the deployment history.
- Programmatic Destruction: The system must provide a DELETE endpoint (/deployments/{container_name}) to allow for destroying a specific deployment via an API call.

**Use-case Diagram**

The use case diagram for DevOps Butler showcases the interactions between the primary user role, the Developer, and the system's core functionalities. The central element is the 'Deploy Application' use case, which serves as the starting point for the entire automated deployment pipeline and is the primary interaction for the developer.

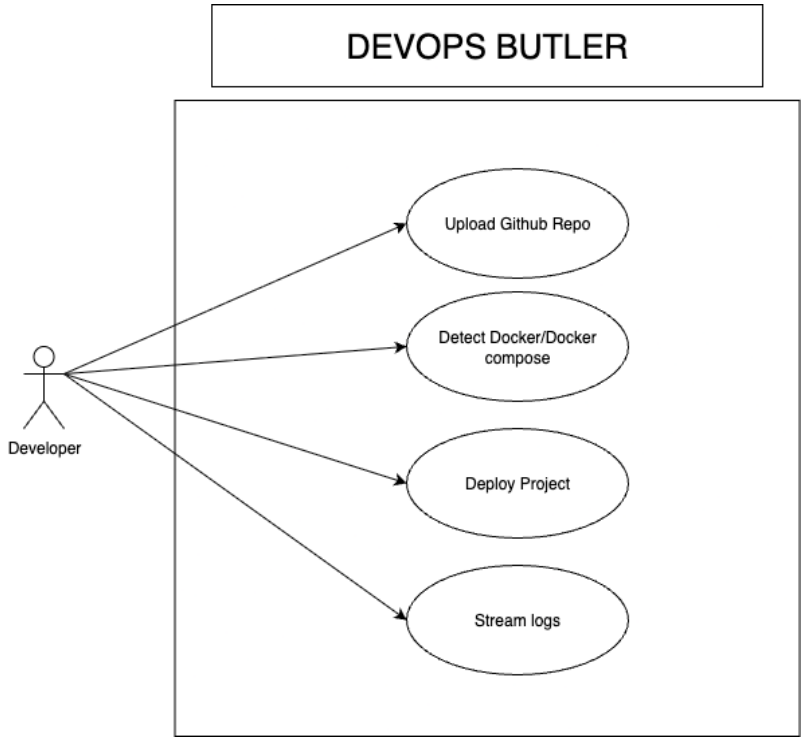



**Figure 3.1: Use-case Diagram of DevOps Butler**

The Developer can utilize a variety of features through the platform to manage the entire application deployment lifecycle. They can initiate a deployment by providing a Git repository URL, monitor the process in real-time by viewing live-streamed logs, and access a dashboard of all currently running applications. The Developer also has a full range of management responsibilities, including viewing the persistent history of all past deployments, destroying a specific application to free up resources, and clearing the entire deployment history. This diagram highlights the comprehensive and self-contained role of the Developer, showcasing how they interact with the system to both initiate automated workflows and manage the platform's core operations and resources from start to finish.

**ii. Non-Functional Requirements**

Non-functional requirements refer to the criteria that are not directly related to the system's functionality but are equally important for the system's success. Non-functional requirements are usually more technical in nature and define how the system should perform or behave. Some non-functional requirements of DevOps Butler are:

a) Reliability and Robustness: The system must be reliable, featuring idempotent deployments that automatically clean up previous versions to prevent conflicts. It must also handle failures gracefully and ensure complete cleanup of stale containers and resources.
b) Usability and Developer Experience: The system must provide a "zero-configuration" experience through a modern, beautiful, and responsive web interface. It should be intuitive, offering one-click deployments to minimize the developer's cognitive load.
c) Performance and Responsiveness: The system must provide high performance, utilizing an asyncio-based framework (FastAPI) to handle long-running tasks. It must deliver real-time feedback with low latency through WebSocket-based log streaming.
d) Resource Management: The platform must manage resources efficiently, using a dedicated Docker network for isolation and providing complete destroy functionality to prevent orphaned containers or resource leaks.
e) Portability and Ease of Setup: The system is designed to run on a local machine with common prerequisites (Python, Docker). The setup process must be simple and well-documented.

### 3.1.2. Feasibility Analysis

The feasibility analysis evaluates whether DevOps Butler can be practically developed and deployed within available technical, operational, economic, and time-based constraints. It helps determine if the project can meet its goals of automating containerized deployments effectively while minimizing risks. The system integrates core functionalities such as repository cloning, project analysis, container building, and real-time logging, making it essential to assess its feasibility from all perspectives.

i. **Technical**

Technical feasibility centers on the available technologies and to what extent they can support the proposed system. According to the feasibility analysis, the technical requirements for DevOps Butler, such as the backend framework, containerization engine, and communication protocols, are all met with mature and widely-used technologies. The system offers a high degree of automation and a user-friendly interface, significantly speeding up the deployment process compared to manual command-line operations. Since the core logic relies on standard Python libraries and Docker commands, the system is technically sound. Technologies are as follows:

- Backend Framework: FastAPI with Python, asyncio, and subprocess.
- Frontend: Vanilla HTML5, CSS3, and JavaScript.
- Real-time Communication: WebSockets.
- Database: SQLite with SQLModel ORM.
- Containerization: Docker & Docker Compose.

ii. **Operational**

Operational feasibility refers to the ability of DevOps Butler to function effectively within a developer's existing workflow and environment. It encompasses a well-defined operational pipeline that addresses the core need for simplified, repeatable deployments. The README.md provides clear, step-by-step instructions for installation and usage, confirming that a user with the specified prerequisites (Python, Docker Desktop) can successfully operate the platform. By automating a complex and error-prone task, DevOps Butler enhances operational efficiency and is highly likely to be adopted and used effectively by its target audience of developers.

iii. **Economic**

Economic feasibility involves assessing the financial implications of implementing the platform. Known as a cost/benefit analysis, this process evaluates the expected benefits against the costs of development and operation. For DevOps Butler, the economic feasibility is highly favorable. The implementation relies entirely on open-source

software (Python, FastAPI, Docker, SQLite), meaning there are no direct licensing costs. The primary benefit is a significant saving in developer time and effort, which translates directly into increased productivity and reduced operational overhead. The project does not necessitate additional hardware beyond a standard development machine, providing a strong economic justification.

iv. **Schedule**

The DevOps Butler project was developed using a milestone-based approach, as evidenced by the "Future Roadmap" section in the README.md. This section shows a clear progression, with key modules such as "Build a Proper Frontend," "Persistence," "Cleanup Logic," and "Enhanced Log Streaming" implemented and marked as complete. This incremental implementation of core features demonstrates that the project's scope was managed effectively. The ability to deliver these foundational components proves that the project was both achievable and manageable within a structured, iterative development schedule, demonstrating strong schedule feasibility.

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

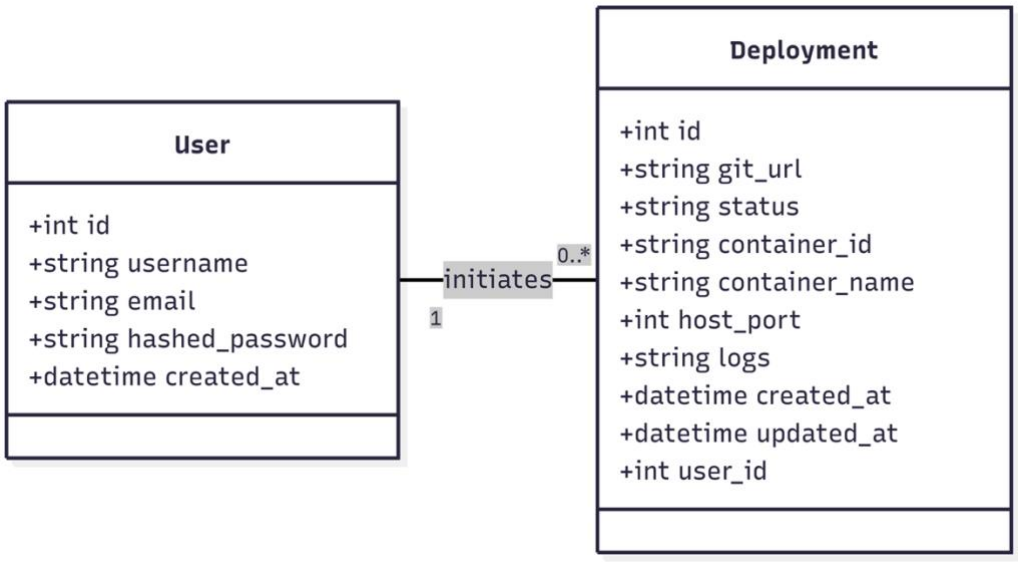



**Figure 3.2: Class Diagram**

This class diagram illustrates the core structure of the DevOps Butler system by showing how its different entities interact. A single User acts as the initiator of all deployment activities and is connected to the Deployment entity. Each Deployment record is created by a User and serves as the central log for a single deployment task, capturing critical data points such as the source git_url, the live status, the resulting container_name, and the accessible host_port. It also stores a complete history of the build and run logs for

debugging and auditing purposes. This object-oriented structure clearly represents the flow of information for managing and tracking automated application deployments, linking every containerized application back to a specific deployment task and the user who initiated it.

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

- **State Diagrams**

  State diagrams visualize how an object's state changes in response to events and conditions, allowing designers and developers to understand and represent complex behavioral logic in a clear and organized manner.

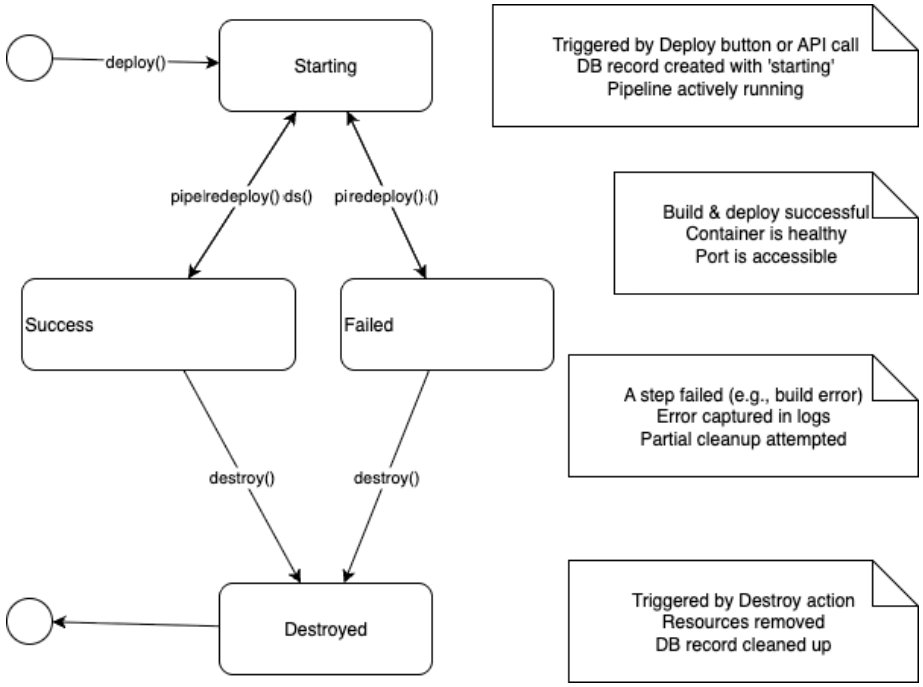



**Figure 3.3: State Diagram**

This state diagram clearly outlines the lifecycle of a deployment within the DevOps Butler System, starting from its initiation in the Starting state to its final outcome as Success, Failed, or Destroyed. It highlights the transitions triggered by user actions such as deploy() and destroy(), as well as internal system events like pipelineSucceeds() and pipelineFails(). Each state—Starting, Success, Failed, and Destroyed—is annotated with relevant notes explaining the pipeline's operational status, the container's health and accessibility, and the state of associated system resources. The diagram provides a concise yet comprehensive visualization of how a deployment progresses through various stages based on user actions and the automated pipeline's outcome.

- **Sequence Diagram**

  Sequence Diagrams are interaction diagrams that detail how operations are carried out. It describes interactions among classes in terms of an exchange of messages over time.

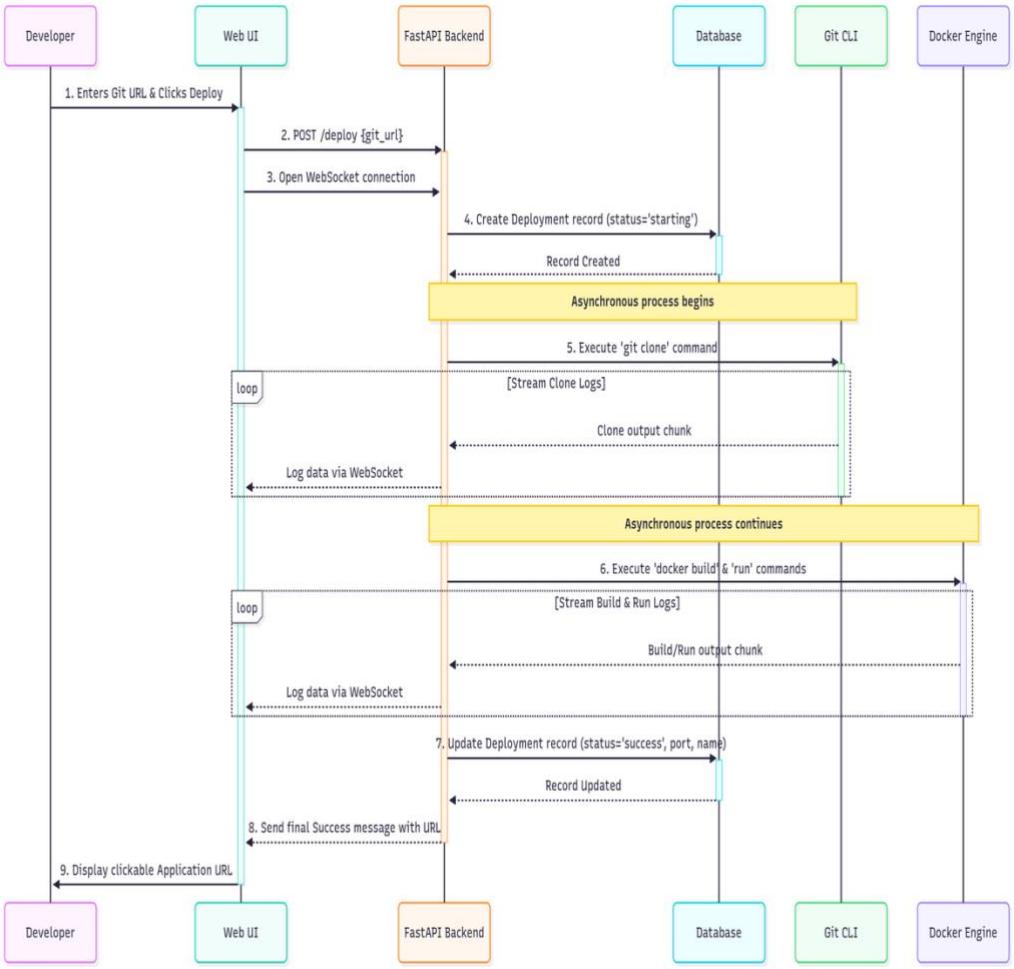



**Figure 3.4: Sequence Diagram**

This sequence diagram effectively illustrates the step-by-step interactions involved in the automated deployment pipeline within DevOps Butler. It shows how the Developer initiates the process by submitting a Git URL through the Web UI, which triggers the FastAPI Backend to orchestrate the entire workflow. The Backend handles internal operations like creating database records and executing asynchronous commands against external tools like the Git CLI and Docker Engine, while streaming real-time logs back to the user. The System then ensures the application is successfully containerized and its accessible URL is returned to the Developer. Distinct participants (Developer, Web UI, FastAPI Backend, Database, Git CLI, and Docker Engine) and clear message flows offer

an organized view of the process, making the dynamic behavior of an automated deployment easy to understand over time.

### 3.1.5. Process Modelling using Activity Diagrams

An Activity Diagram visually traces the step-by-step execution of processes or operations within a system. It functions like a flowchart, illustrating the sequence of actions and decisions that occur. This diagram is crucial for understanding the dynamic aspects of a system, showing how control flows from one activity to the next. Essentially, it maps out the operational steps a system takes to achieve a specific outcome

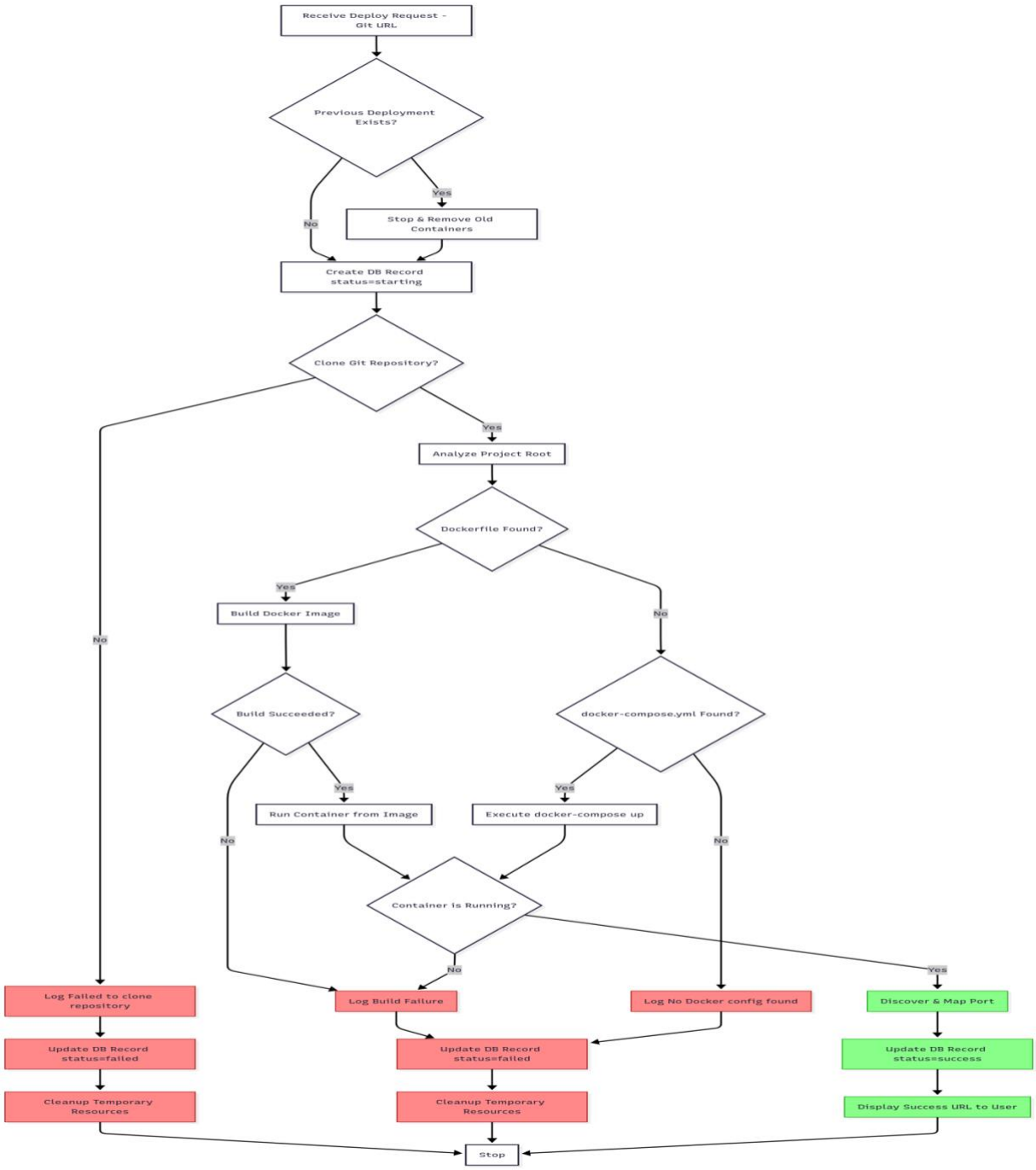



**Figure 3.5 Activity Diagram**

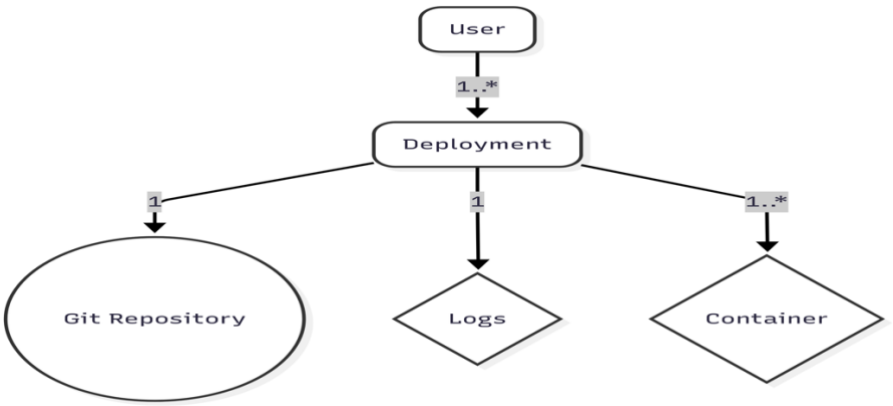
This activity diagram illustrates how DevOps Butler processes an automated deployment request from a developer. The process begins when the developer submits a Git URL; upon receiving the request, the system first cleans up any pre-existing deployments of the same repository to ensure a clean state. From there, the system clones the repository and analyzes its structure to find a Dockerfile or docker-compose.yml. Based on this analysis, it chooses a deployment strategy and proceeds to build and run the container. If the pipeline is successful, the application's port is mapped, the database is updated to 'success', and the final URL is displayed to the developer. If any step in the process fails, the workflow is diverted to a failure state where the error is logged and resources are cleaned up. This diagram visualizes the complete, step-by-step automated workflow of the deployment pipeline, including its decision points and failure-handling logic.

## 3.2. System Design

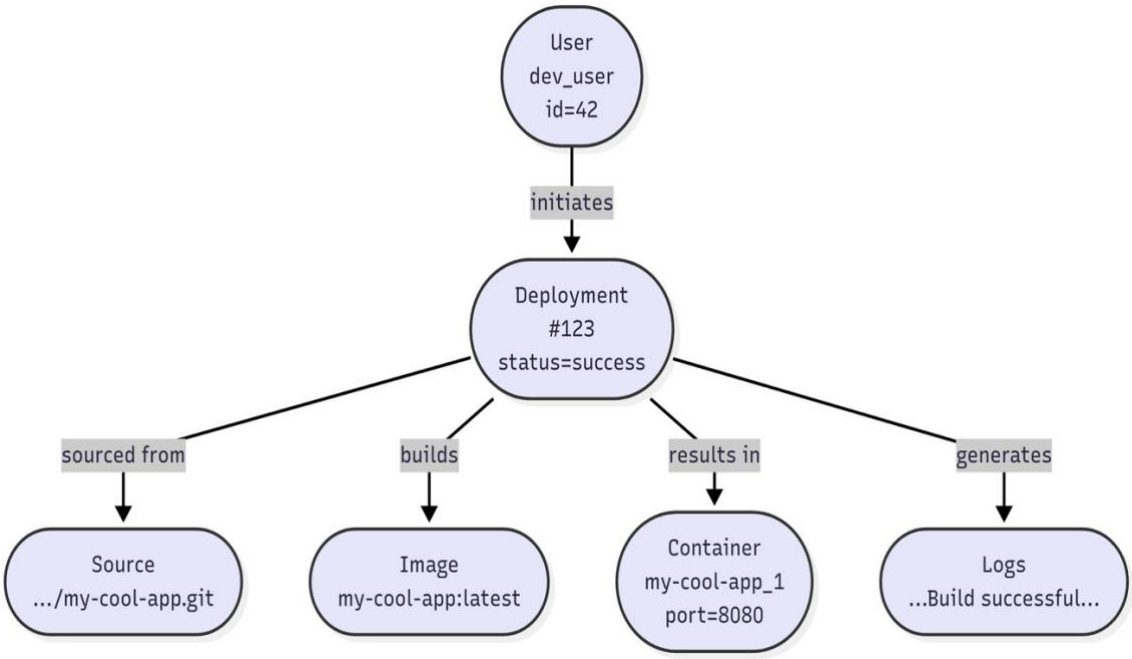
In the DevOps Butler system, users interact mainly as Developers. Developers can initiate automated deployments from public Git repositories, monitor the entire build and run process via live-streamed logs, and manage their running applications. Each deployment is recorded in a Deployment table, which holds references to the source Git repository, the current status of the pipeline, and the final container details like its name and mapped port. The system's orchestration engine manages the entire deployment pipeline, executing commands against external tools like the Git CLI and the Docker Engine. Real-time interaction is maintained via a WebSocket connection, which streams logs and status updates from the backend directly to the developer's interface. Every part of the system, from the initial repository analysis to the final container lifecycle management, is interconnected to ensure a reliable, automated, and user-friendly deployment platform.

### 3.2.1. Refinement of Class, Object, State, Sequence and Activity Diagrams

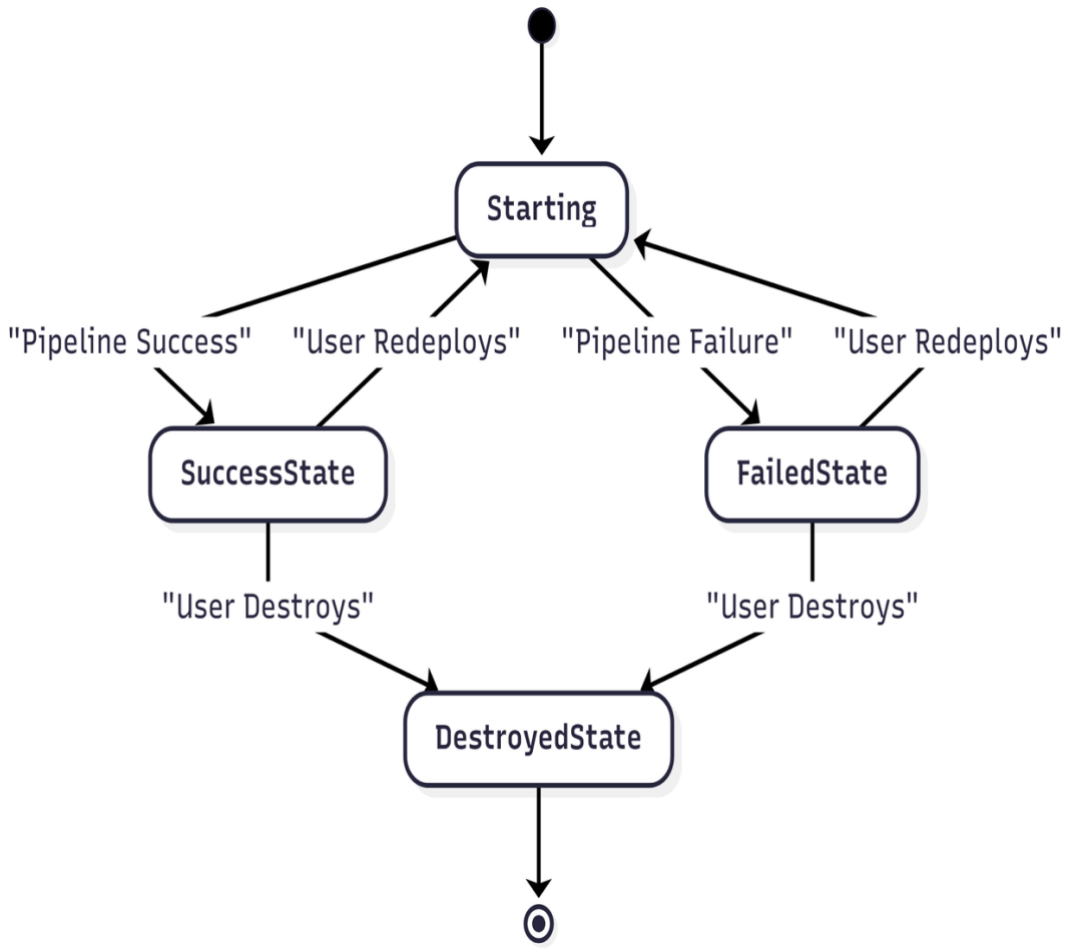
**Class**



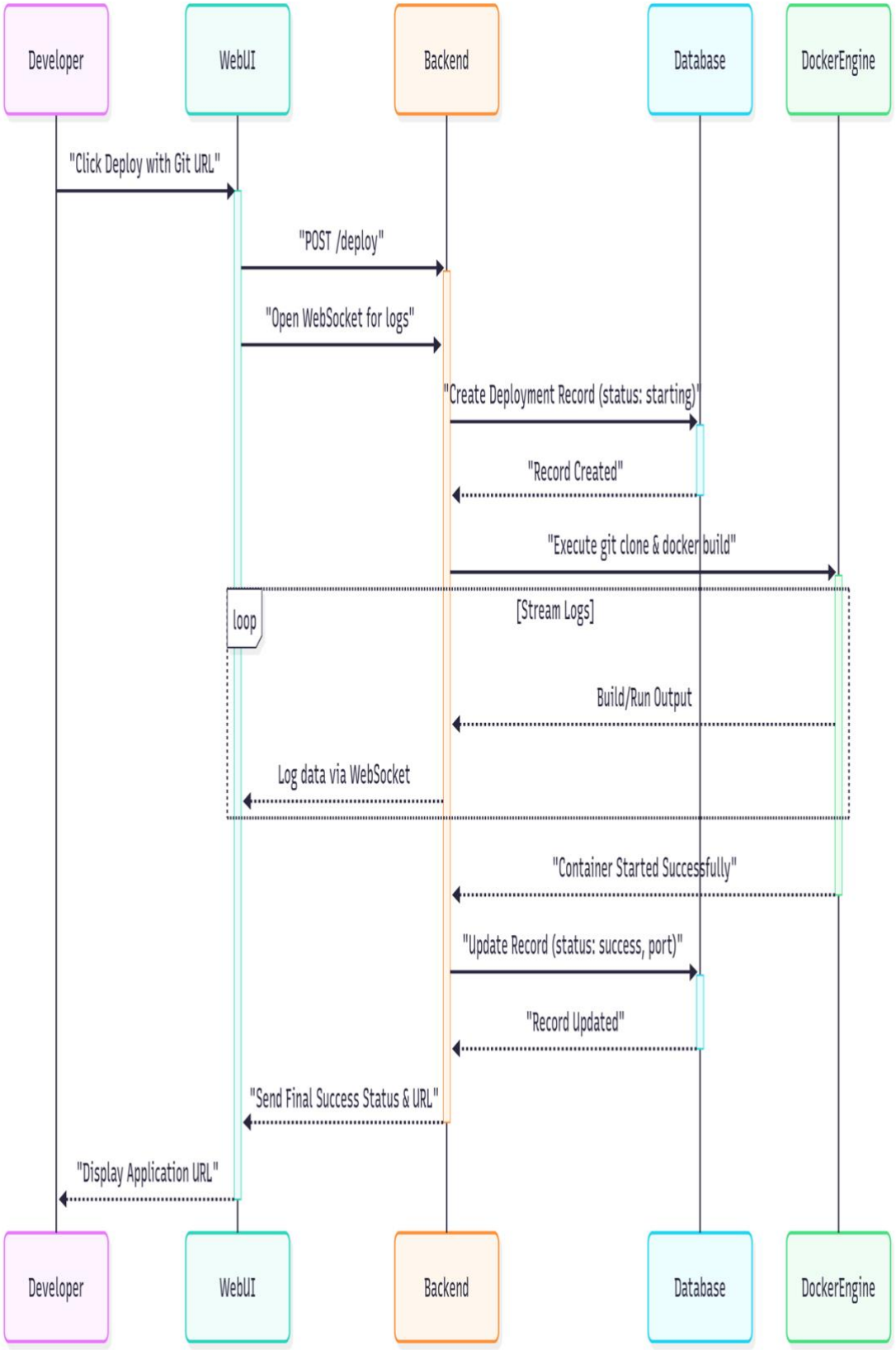
**Object**




**State**



**Sequence**


Developer
WebUI
Backend
Database
DockerEngine
"Click Deploy with Git URL"
"POST /deploy"
"Open WebSocket for logs"
"Create Deployment Record (status: starting)"
"Record Created"
"Execute git clone & docker build"
loop
[Stream Logs]
Build/Run Output
Log data via WebSocket
"Container Started Successfully"
"Update Record (status: success, port)"
"Record Updated"
"Send Final Success Status & URL"
"Display Application URL"
Developer
WebUI
Backend
Database
DockerEngine

**Activity**

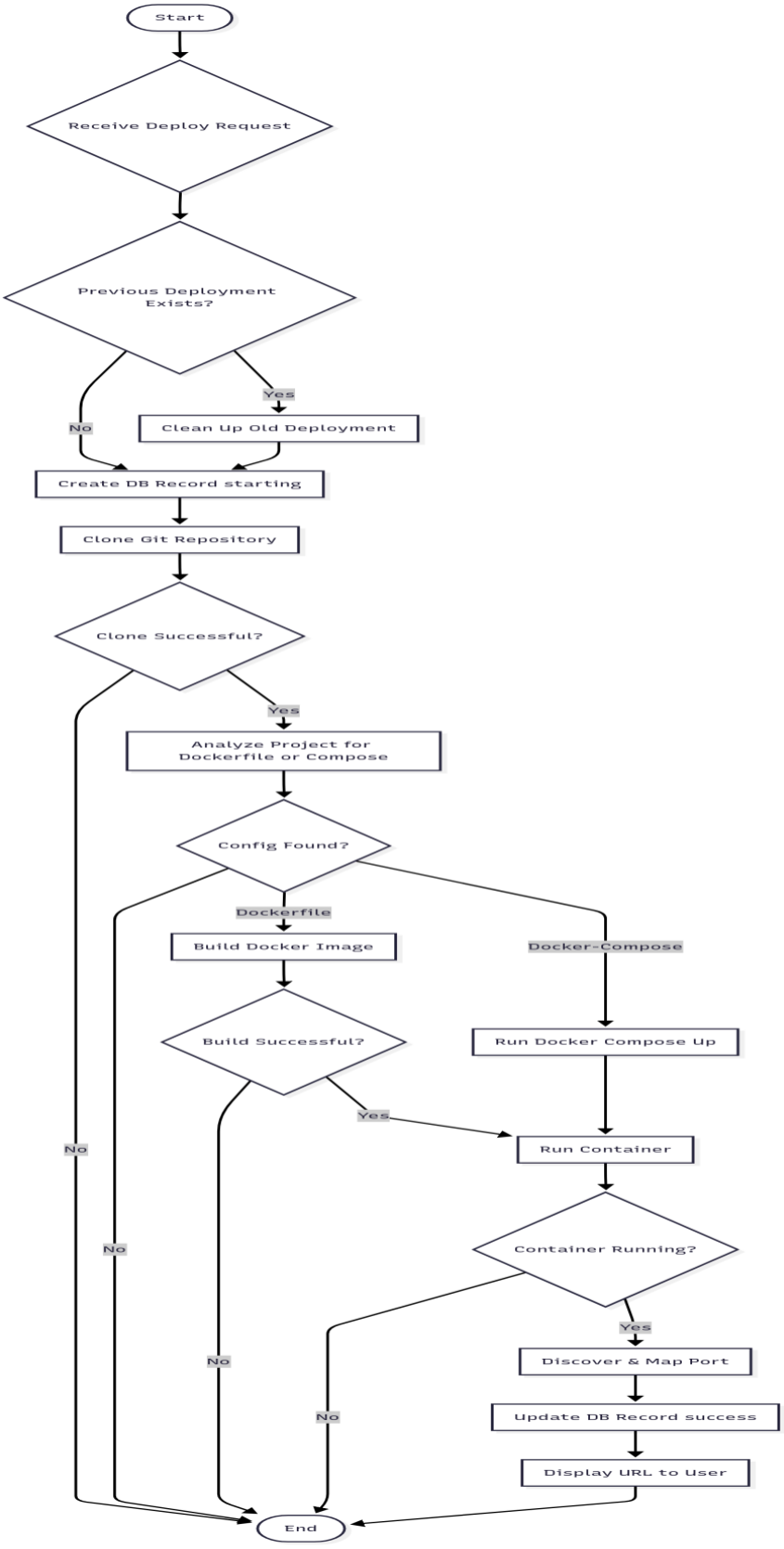



**Figure 3.6 Refinement of Class, Object, State, Sequence and Activity Diagrams**

### 3.2.2. Component Diagrams

The Component Diagram for DevOps Butler provides a high-level view of the system's structure by breaking it into modular components and showing how they interact. It includes major parts such as the Web Interface (Frontend), where developers initiate deployments, view live logs, and manage applications; the Orchestration Engine (Backend), which contains the core functionalities like repository analysis, pipeline execution, and container lifecycle management; and the Data & External Services Layer, which consists of the SQLite database for persistence, the WebSocket service for real-time communication, and the interfaces to external command-line tools like Docker and Git. These components are interconnected to support the automated deployment workflow—developers interact with the Web Interface to trigger actions, the Orchestration Engine processes these requests by commanding external tools, and all modules communicate with the Database for state storage and retrieval. This diagram is useful for understanding the system architecture, planning development tasks, and ensuring clear communication between the user-facing, logical, and execution layers of the platform.

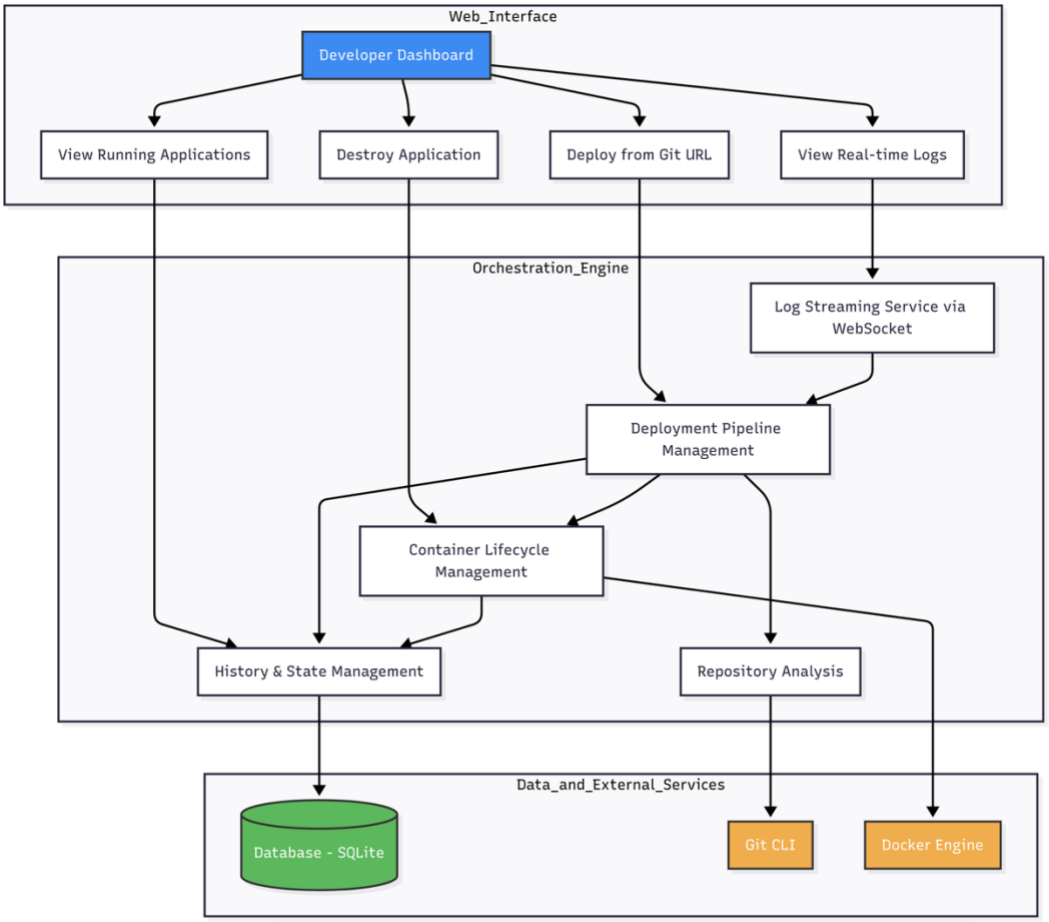



**Figure 3.7 Component Diagrams**

### 3.2.3. Deployment Diagrams

The Deployment Diagram for the Sazlo Course Selling Site illustrates the physical architecture of the system, showing how software and hardware components are connected in a real-world environment. It includes components like User Devices (used by students and admins), a Web Server that hosts the frontend, an Application Server for handling business logic, a Database Server for storing data, and external services like the Payment Gateway and Notification Service. These components are typically hosted in the cloud, and they communicate with each other to support the full functionality of the platform — from browsing and buying courses to receiving notifications and processing payments. This diagram helps in understanding the system infrastructure, server interactions, and external dependencies involved in deploying the application.

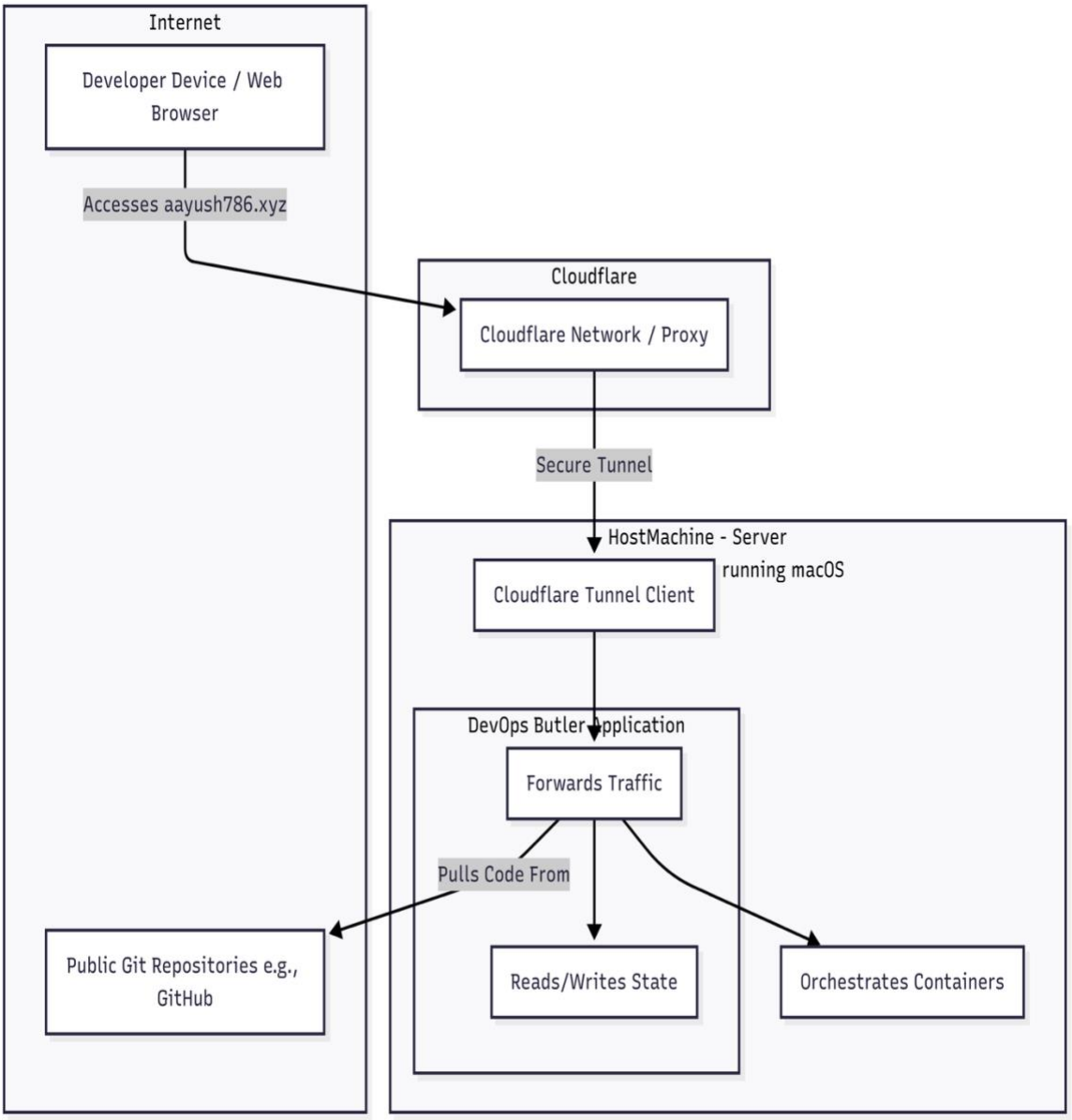



**Figure 3.8 Deployment Diagrams**

## 3.3. Algorithm Details

### 1. Deployment Pipeline Orchestration Algorithm

Purpose: To provide a fully automated, end-to-end process for turning a Git repository into a running containerized application.

How it works: The algorithm follows a strict, sequential flow of operations:

- Validate the incoming Git URL.
- Execute the Idempotent Cleanup Algorithm to remove any previous deployments.
- Create a database record with a "starting" status.
- Clone the Git repository.
- Execute the Repository Analysis Algorithm to select a deployment strategy.
- Build/run the container(s) based on the selected strategy.
- Discover the container's mapped host port.
- Update the database record with the final status ("success" or "failed").

Use in Project: This is the central algorithm that orchestrates the entire system.

### 2. Repository Analysis Algorithm

Purpose: To intelligently and automatically determine the correct method for building and running the cloned repository.

How it works (Multi-level Detection):

- URL Validation: Uses pattern matching to ensure the provided string is a valid Git repository URL and not from an unsupported source like a container registry.
- File Structure Analysis: Performs a directory traversal of the repository's root to detect the presence of docker-compose.yml or Dockerfile to select the deployment strategy.

Use in Project: A critical step within the main pipeline, executed immediately after cloning.

### 3. Container Lifecycle Management Algorithm

Purpose: To manage the full state of containers, from creation to destruction, ensuring predictable behavior and clean resource management.

How it works (State Management):

- Container State Tracking: Monitors the status of deployments, storing state (e.g., 'starting', 'success', 'failed') in the database.
- Cleanup Algorithm (Idempotency): Before a new deployment begins, it actively stops and removes any old containers associated with the same repository to

prevent conflicts. It also provides a dedicated "destroy" function for complete manual cleanup.

Use in Project: Underpins the platform's reliability, idempotency, and resource management features.

**4. Real-time Communication Algorithm**

Purpose: To provide the developer with immediate, live feedback on the progress of the deployment pipeline.

How it works (WebSocket Broadcasting System):

- Connection Management: Establishes and maintains a WebSocket connection between the frontend and backend for the duration of a deployment.
- Asynchronous Message Dispatching: As the backend executes long-running build commands, it captures stdout/stderr in real-time and asynchronously pushes each line of output as a message through the WebSocket to the connected client.

Use in Project: This algorithm powers the "live, color-coded logs" feature, a core part of the developer experience.

# Chapter 4: Implementation and Testing

## 1.1. Implementation

This section talks about the implementation of the system.

### 4.1.1. Tools Used

**i. CASE Tools**

a. Mermaid.js / Draw.io

It was used to create UML diagrams such as the Use Case, Class, Sequence, State, and Activity diagrams to model the system's structure and behavior.

b. Markdown Editor

It was used for the purpose of documenting the entire project, including the README.md file, which outlines the project's features, architecture, and setup instructions.

**ii. Frameworks & Languages**

a. FastAPI (Python)

FastAPI is a modern, high-performance Python web framework. It was used to build the entire backend of DevOps Butler, including the RESTful API, WebSocket server for real-time logging, and the core orchestration logic for managing deployments.

b. HTML5

It was used to structure the content and layout of the DevOps Butler web interface, defining elements such as the deployment form, the log display area, and the application dashboard.

c. CSS3

It was used to design and control the visual presentation of DevOps Butler, creating a modern, responsive, and visually appealing user interface for a seamless developer experience.

d. JavaScript (Vanilla)

It was used to make the DevOps Butler web interface dynamic and interactive. It handles form submissions, establishes and manages the WebSocket connection for receiving live logs, and updates the UI in real-time based on deployment status.

**iii. Platform & Engine**

a. Docker & Docker Compose

These are the core containerization technologies that DevOps Butler automates. The platform uses Python's subprocess module to execute docker and docker-compose commands to build images, run containers, and manage their lifecycle.

**iv. Database Platform**

a. SQLite

SQLite was used as the lightweight, file-based relational database management system for DevOps Butler. It provides persistent storage for tracking deployment history, status, and container details, managed via the SQLModel ORM. This section talks about the implementation of the system.

### 4.1.2. Implementation Details of Modules

The DevOps Butler platform is designed with several key modules that work together to provide a seamless and automated deployment experience. Each of these modules plays a crucial role in the functionality of the platform. Below is an overview of the most important modules and the core functions they perform.

**1. Deployment Orchestration Module**

a. Deploy Application

This function allows users to initiate a new deployment by providing a public Git repository URL via the web interface or an API call. The system validates the URL, initiates the idempotent cleanup of any previous deployment, and kicks off the entire automated pipeline.

- Function

  Handles the incoming deployment request, validates the Git URL, and initiates the end-to-end deployment pipeline from cloning to running the container.

**2. Repository Analysis Module**

a. Analyze Project Structure

This function is responsible for the "intelligent analysis" of the cloned repository. It scans the project's root directory to automatically detect the presence of a Dockerfile or docker-compose.yml to determine the correct deployment strategy without requiring any user configuration.

- Function

  Scans the root of the cloned repository to detect the deployment configuration and selects the appropriate build and run strategy.

**3. Real-time Logging Module**

a. Stream Deployment Logs

This function uses WebSockets to provide live, color-coded feedback to the user during the deployment process. It captures all output from the cloning, building, and container execution steps and streams it directly to the user's browser in real-time.

- Function

  Establishes a WebSocket connection and streams live output from the backend processes directly to the user's interface, providing full transparency.

**4. Container Lifecycle Management Module**

a. View Running Applications

This function provides visibility into the currently active deployments. It fetches the status and details of all containers managed by the platform and displays them on a dashboard in the web UI.

- Function

  Retrieves and displays a list of all currently running applications, their mapped ports, and health status.

  b. Destroy Deployment

  This function provides complete cleanup functionality. It allows a user to stop and permanently remove a running container and all its associated resources. It also cleans up the corresponding record in the database to prevent conflicts.

- Function

  Handles the complete teardown of a specified deployment, including stopping and removing the Docker container and clearing its database entry.

**5. Deployment History Module**

a. Track and Display History

This function is responsible for the persistent tracking of all deployment activities. It records the status, timestamps, Git URL, and container details for every deployment attempt in the SQLite database and provides an interface to view this history.

- Function

  Persists all deployment records in a database and provides a view and an API endpoint (/deployments) for retrieving this historical data.

**6. User Authentication Module**

a. User Registration

This function allows a new developer to create an account on the platform, as indicated by the /api/auth/register API endpoint.

- Function

  Handles the registration process for new users.

  b. User Login

  This functionality lets an authenticated user access the platform's features, as indicated by the /api/auth/login API endpoint.

- Function

  Authenticates a user's credentials to grant access to the system.

## 4.2. Testing

### 4.2.1. Test Cases for Core Functionality

**Table 4.1 Deployment Pipeline Test Case**

| S.N | Test Case Description | Test Data | Expected Result | Actual Result | Pass /Fail |
|---|---|---|---|---|---|
| 1 | Deploy a valid project with a Dockerfile in the root. | git_url: “https://github.com/user/simple-dockerfile-app.git” | Deployment succeeds. A container is running and accessible via its mapped port. History shows “success”. | As Expected | Pass |
| 2 | Deploy a valid project with a docker-compose.yml in the root. | git_url: “https://github.com/user/simple-compose-app.git” | Deployment succeeds. The compose service is running and accessible. History shows “success”. | As Expected | Pass |
| 3 | Deploy a project with a broken Dockerfile (e.g., invalid command). | git_url: “https://github.com/user/broken-dockerfile-app.git” | Deployment fails. The real-time logs display the Docker build error. No container is left running. History shows “failed”. | As Expected | Pass |
| 4 | Deploy a project with no Dockerfile or docker-compose.yml. | git_url: “https://github.com/user/no-docker-config-app.git” | Deployment fails immediately after cloning with a “No deployment strategy found” message in the logs. | As Expected | Pass |
| 5 | Attempt to deploy using an | git_url: “this-is-not- | System returns an immediate validation | As Expected | Pass |

| | | | | | |
|---|---|---|---|---|---|
| | invalid Git URL format. | a-valid-url” | error. No deployment process is initiated. | | |

This test case verifies the core deployment pipeline of DevOps Butler. It focuses on the system's ability to handle the primary success scenarios (both Dockerfile and Docker Compose) and critical failure scenarios, such as a broken configuration, a missing configuration, or invalid user input.

**Table 4.2 Container Lifecycle Management Test Case**

| S.N | Test Case Description | Test Data | Expected Result | Actual Result | Pass/Fail |
|---|---|---|---|---|---|
| 1 | Destroy a successfully running deployment. | container_name: The name of the container from Test Case 1. | The container is stopped and removed from the Docker engine. The application is no longer listed on the dashboard. | As Expected | Pass |
| 2 | Redeploy an already deployed repository (Idempotency Check). | git_url: The same URL from Test Case 1. | The previously running container is automatically stopped and removed. The new deployment proceeds and succeeds without port conflicts. | As Expected | Pass |
| 3 | Attempt to destroy a non-existent deployment. | container_name: “non-existent-container” | The system returns an error message stating the container was not found, without crashing. | As Expected | Pass |

This test case covers the crucial lifecycle management features of the platform. It ensures that users can cleanly remove applications and that the system's idempotency logic correctly handles redeployments to prevent resource conflicts, which is vital for a stable deployment environment.

**Table 4.3 User Interface and Real-time Logging Test Case**

| S. N | Test Case Description | Test Data | Expected Result | Actual Result | Pass/ Fail |
|---|---|---|---|---|---|
| 1 | Observe the log stream during a multi-step build process. | A deployment with a Dockerfile that has several RUN commands. | Log lines appear in the UI's log viewer in real-time as each step executes, not all at once at the end. | As Expected | Pass |
| 2 | Verify the final application URL is displayed correctly after success. | A successful deployment. | The UI displays a clickable link in the format http://<container_name>.localhost:<host_port>. | As Expected | Pass |
| 3 | Check that the deployment history updates after a deployment. | Any deployment attempt (success or fail). | A new entry appears at the top of the deployment history list with the correct Git URL, status, and timestamp. | As Expected | Pass |

This test case evaluates the developer experience provided by the DevOps Butler UI. The tests confirm that the real-time logging is functional and provides immediate feedback, and that the UI accurately reflects the status and outcome of each deployment, from the final URL to the persistent history log.

**Table 4.4 User Authentication Test Case**

| S.N | Test Case Description | Test Data | Expected Result | Actual Result | Pass/Fail |
|---|---|---|---|---|---|
| 1 | Register a new user with valid details. | email: “dev@example.com”, password: “securepass123” | User account is created successfully. A success token or message is returned. | As Expected | Pass |
| 2 | Log in with correct credentials. | email: “dev@example.com”, password: “securepass123” | Login is successful. An authentication token is provided to access protected routes. | As Expected | Pass |
| 3 | Attempt to log in with an incorrect password. | email: “dev@example.com”, password: “wrongpassword” | Login fails. An “Invalid credentials” error message is returned. | As Expected | Pass |

The User Authentication Test Cases assess the functionality of the user management system as defined by the API endpoints. These tests verify that new users can register, existing users can securely log in, and that the system properly handles authentication failures, which is fundamental for any multi-user platform.

## 4.3. Result Analysis

The system testing phase for DevOps Butler was conducted to validate the functionality, reliability, and usability of the platform from a developer's perspective. A total of 14 test cases were executed, covering critical functionalities such as the core deployment pipeline for both Dockerfile and docker-compose.yml projects, container lifecycle

management including idempotency and cleanup, the real-time logging user interface, and basic user authentication.

**Table 4.5 Result Analysis**

| Test Case | Description | Result |
|---|---|---|
| 1 | Deploy with valid Dockerfile | Pass |
| 2 | Deploy with valid docker-compose.yml | Pass |
| 3 | Deploy with broken Dockerfile (Error Handling) | Pass |
| 4 | Deploy with no Docker configuration | Pass |
| 5 | Deploy with invalid Git URL (Validation) | Pass |
| 6 | Destroy a running deployment | Pass |
| 7 | Redeploy an existing repository (Idempotency) | Pass |
| 8 | Destroy a non-existent deployment | Pass |
| 9 | Real-time log streaming | Pass |
| 10 | Display final success URL in UI | Pass |
| 11 | Deployment history updates correctly | Pass |
| 12 | User Registration | Pass |
| 13 | User Login (Successful) | Pass |
| 14 | User Login (Failed) | Pass |

# Chapter 5: Conclusion and Future Recommendations

## 5.1. Conclusion

In conclusion, the development of DevOps Butler has resulted in a sophisticated, functional, and highly intuitive automated deployment assistant, specifically tailored to the needs of modern developers seeking a "zero-configuration" experience. The application successfully achieves its core mission of transforming a simple Git repository URL into a fully containerized, running application with a single click. The platform effectively incorporates a powerful set of features, including intelligent repository analysis for Dockerfile and docker-compose.yml configurations, complete container lifecycle management, real-time log streaming via WebSockets, and idempotent deployments for robust, conflict-free operations. The entire system is built upon a modern and high-performance tech stack, with a FastAPI backend orchestrating the workflow and a clean, responsive frontend built with vanilla web technologies.

The modular and event-driven architecture facilitated a focused and iterative development process, allowing for the successful implementation of foundational features like database persistence and a complete "destroy" functionality for resource cleanup. By abstracting

away the complexities of manual Docker commands, port mapping, and network management, the project has not only met its initial objectives of simplifying the deployment pipeline but has also established a superior developer experience that prioritizes speed, transparency, and ease of use. Overall, DevOps Butler stands as a powerful proof-of-concept and a solid foundation upon which a more comprehensive and feature-rich Platform-as-a-Service (PaaS) can be built.

## 5.2. Future Recommendation

As it stands, the DevOps Butler system successfully fulfills its core vision of providing a seamless local deployment experience. However, as with any ambitious platform, no system is perfect, and there is a clear and exciting roadmap for future enhancements that will elevate it from a powerful local tool to a true, production-ready PaaS. The following recommendations are based on the project's official future roadmap:

1) **Implement Multi-Environment Support**: A critical next step is to introduce support for distinct deployment environments, such as staging and production. This would allow developers to manage different versions of their applications, test changes in an isolated staging environment before promoting them to production, and manage environment-specific configurations and variables.
2) **Integrate Automatic Health Checks**: To improve reliability, the system should be enhanced with automatic health monitoring for all deployed applications. This would involve periodically checking defined health endpoints on the containers to ensure they are responsive. If an application becomes unhealthy, the system could be configured to automatically restart it or alert the developer, adding a layer of self-healing capability.
3) **Introduce Resource Monitoring**: Providing developers with visibility into their application's performance is crucial. Future versions should incorporate resource monitoring to track and display real-time CPU, memory, and disk usage for each running container. This data would be invaluable for debugging performance issues and optimizing applications.
4) **Add Support for Custom Domains**: To make the platform viable for public-facing applications, support for custom domains is essential. This feature would allow developers to easily map their own domain names (e.g., my-app.com) to their deployed services, moving beyond the default localhost-based URLs.

5) **Enable Automatic SSL/TLS Support**: In conjunction with custom domains, the platform should integrate automatic HTTPS certificate generation and renewal, for instance, by using Let's Encrypt. This would ensure that all deployed applications are served securely over HTTPS by default, a standard requirement for modern web applications.
6) **Transform into a Cloud-Native PaaS**: The ultimate goal is to evolve DevOps Butler into a true Platform-as-a-Service capable of deploying applications to the cloud. This would involve significant architectural enhancements to support cloud provider integrations (AWS, GCP, Azure), enabling developers to deploy their applications to scalable, production-grade infrastructure directly from the Butler interface.

# References

[1] [1 Phillips, R. (2019). Learning Management Systems: Choosing the Right One for Your Organization. Association for Talent Development.

[2] Morrison, D. (2018). Designing Effective Learning Management Systems:

[3] Instructional Design Strategies for Today's Educators. Routledge.

[4] Pappas, C. (2020). The Complete Guide to Learning Management Systems. ELearning Industry.

[5] Adams, J., & Stein, J. (2017). Learning Management System Technologies and Software Solutions for Online Teaching: Tools and Applications. IGI Global.

[6] Terrell, S. R. (2015). Designing and Implementing Effective Online Learning Environments. Springer.

[7] Watson, W. R., & Watson, S. L. (2017). Digital Learning: Strengthening and Assessing 21st Century Skills, Grades 5-8. Corwin.

[8] Rieber, L. P. (2016). Enhancing Learning Through Technology: Research on Emerging Technologies and Pedagogies. Routledge.

[9] West, R. E., & Williams, G. S. (2018). Technology and Pedagogy in the Digital Classroom: Developing New Teaching and Learning Practices. IGI Global.

[10] McGonagle, J. J., & Vella, K. (2018). Content Marketing: A Practical Guide for Startups. Kogan Page.

[11] Nielsen, J. (2016). Designing Web Usability: The Practice of Simplicity. New Riders Publishing