VISVESVARAYA TECHNOLOGICAL UNIVERSITY

BELAGAVI, KARNATAKA-590 018

MINI PROJECT REPORT ON

"Custom PaaS: DockLaunch"

Submitted in partial fulfilment of the requirements for the course Cloud Computing (21CS72)

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

By

AKRITY KUMARI GUPTA (1JS21CS018)
ALOK KUMAR MAURYA (1JS21CS019)
KUMARI SANYA RAI (1JS21CS079)

Under the guidance of
Dr. Sharana Basavana Gowda
Associate Professor, Dept of CSE,
JSSATE, Bengaluru

JSS ACADEMY OF TECHNICAL EDUCATION, BENGALURU
Department of Computer Science and Engineering
2024 – 2025
JSS MAHAVIDYAPEETHA, MYSURU

JSS Academy of Technical Education

JSS Campus, Uttarahalli Kengeri Main Road, Bengaluru – 560060

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that A MINI PROJECT REPORT entitled **“Custom PaaS: DockLaunch”** has successfully carried out by **Akrity Kumari Gupta (1JS21CS018), Alok Kumar Maurya (1JS21CS019), and Kumari Sanya Rai (1JS21CS079)** in partial fulfilment for the course Cloud Computing (21CS72) of 8th Semester **Bachelor of Engineering in Computer Science and Engineering in Visvesvaraya Technological University Belagavi** during the year 2024-2025. It is certified that all corrections and suggestions indicated for Internal Assessment have been incorporated in the report deposited in the department library. The mini project report has been approved as it satisfies the academic requirement in respect of the project work prescribed for the said degree.

Dr. Sharana Basavana Gowda
Associate Professor,
Dept. of CSE,
JSSATE, Bengaluru

ABSTRACT

In today’s fast-paced software development environment, the need for efficient deployment solutions is paramount. This project presents a Deployment Management System that automates the process of deploying applications from GitHub repositories into Docker containers and exposes them to the public internet using Ngrok. The system simplifies the deployment workflow by handling tasks such as cloning a repository, building a Docker image, running the container, and providing a publicly accessible URL through Ngrok.

The system is designed for ease of use, where the user only needs to provide a GitHub repository URL. The backend processes ensure the repository is cloned, a Docker image is built, and the application is deployed in a container, with a public URL generated via Ngrok for easy access. The system also supports container and Ngrok process cleanup after the deployment, ensuring no residual resources are left behind.

This solution streamlines the deployment process, making it easier for developers to test and share their applications without configuring complex infrastructure or networks. With future scalability in mind, the system can be enhanced to support multi-repository deployments, user authentication, and integration with cloud services. Ultimately, this project provides a foundation for continuous deployment automation in software development, reducing manual effort and facilitating rapid application testing and sharing.

INDEX

Chapter No.	Topic Name	Page No.
1.	Introduction	3
2.	System Analysis	4
3.	Methodology	8
4.	Result	10
5.	Future Scope	16
6.	Conclusion	17
7.	References	18

LIST OF FIGURES

Figure No.	Figure Name	Page No.
1.	Landing Page for DockLaunch	12
2.	Feature Section on Landing Page	13
3.	Sample React Project Repository	13
4.	Deploying a Application via Github repo link	14
5.	Application Deployed Successfully with Active Status	14
6.	Hosted Application can now be accessed with Public url	15
7.	Stopping the Application	15

1. INTRODUCTION

In the modern era of software development, continuous deployment and rapid application deployment are crucial for developers to streamline their workflows and minimize the complexity of setting up environments. This report describes the design, implementation, and results of a Deployment Management System that automates the process of deploying GitHub repositories into Docker containers and exposing them to the public internet using Ngrok. The system aims to simplify the deployment process for developers, allowing them to deploy and access their applications with minimal configuration.

The main objectives of the system are:

- Automating the process of cloning repositories from GitHub.
- Building and running the application in a Docker container.
- Exposing the application to the public internet via Ngrok.
- Providing users with a public URL to access their deployed applications.

This deployment management system is designed to provide a seamless and secure environment for testing web applications without the need to configure complex infrastructure manually.

2. SYSTEM ANALYSIS

The **Deployment Management System** is designed to automate and streamline the deployment process by leveraging GitHub, Docker, and Ngrok. Below is a detailed breakdown of the system's components and requirements:

Key Components:

1. **GitHub Repository:**
   - The GitHub repository serves as the origin of the application code. The user provides the URL of the repository they wish to deploy.
   - The system interacts with GitHub using the repository URL to clone the code into a local directory for further processing.
   - The repository can contain various types of applications such as web applications, APIs, or microservices, along with their dependencies and configuration files.
2. **Docker:**
   - **Docker** is a containerization platform that encapsulates an application along with its dependencies, environment settings, and runtime in a container. Containers ensure that applications run consistently across different environments, providing isolation from the host system and other applications.
   - The system uses Docker to:
     - **Build a Docker image** from the repository. This image contains all the necessary components to run the application.
     - **Run the Docker container** from the built image. The container runs the application in a lightweight, isolated environment, ensuring it functions the same way across different systems.
   - Docker handles complexities related to dependencies, operating system compatibility, and application setup, ensuring that the application runs as expected without external configuration.
3. **Ngrok:**
   - **Ngrok** is a service that allows developers to expose their local applications to the public internet through a secure tunnel. This eliminates the need for setting up complex networking configurations like port forwarding or cloud infrastructure.
   - In the system, Ngrok is used to:
     - **Generate a public URL** for the Docker container, making the application accessible over the internet.

        - The system automatically starts an Ngrok process, which assigns a secure HTTPS URL to the application running in the Docker container.
        - This feature is especially useful for testing applications, demoing prototypes, or providing temporary access to applications hosted locally, without requiring users to deal with the intricacies of server configurations.

4. **User Input:**
    - The system is designed to be intuitive and easy to use. The user's primary task is to provide the **GitHub repository URL** of the application they wish to deploy.
    - Once the repository URL is provided, the system automates the rest of the deployment process, including:
        - Cloning the repository to the local server.
        - Building a Docker image from the repository.
        - Running the Docker container.
        - Starting the Ngrok process and generating a public URL for the application.
    - The simplicity of this input (just a URL) hides the underlying complexities of repository management, containerization, and public exposure of the application.

System Requirements:

For the system to function correctly, the following tools must be installed and configured on the host machine:

1. **Docker:** Docker must be installed and running on the system. It is essential for containerizing the application and isolating its environment. Docker must also be properly configured to allow the system to interact with the Docker daemon and perform operations like building images and running containers.
2. **Ngrok:** Ngrok must be installed to expose local applications to the internet. The system uses Ngrok to generate a public URL that allows users to access their deployed applications over HTTPS. Ngrok is a lightweight tool that works well in development environments where there is no need to set up a full-fledged server.
3. **Git:** Git must be installed to enable the system to clone the repository from GitHub. Git allows the system to pull the application code from the specified repository URL, which is then used to build the Docker image.
4. **Python Libraries:** The system uses various Python libraries, such as:

- requests: For handling HTTP requests to interact with the Ngrok API and retrieve the public URL.
- docker: For interacting with Docker and managing containers and images.
- gitpython: For cloning the GitHub repository.

System Design and Workflow:

The system is designed to automate the deployment process, handling all the technical complexities in the backend. Here's an overview of the workflow:

1. **User Input:**
   - The user inputs the GitHub repository URL into the system.
   - The system uses this URL to begin the process of cloning the repository to the local machine.
2. **Cloning the Repository:**
   - The system uses Git to clone the repository into a designated folder on the local server.
   - If the repository already exists locally, the system skips this step.
3. **Building the Docker Image:**
   - The system checks for a Dockerfile in the cloned repository. If present, it proceeds to build a Docker image from the repository's code.
   - The image is tagged with a unique identifier based on the repository name.
4. **Running the Docker Container:**
   - Once the image is built, the system runs it as a container, exposing the necessary ports for the application to function.
   - The application is now running in an isolated environment managed by Docker.
5. **Starting Ngrok:**
   - The system automatically starts Ngrok, which creates a secure tunnel from the local machine to the public internet.
   - Ngrok generates a public HTTPS URL that points to the application running in the Docker container.
6. **Accessing the Application:**
   - The system returns the Ngrok URL to the user, allowing them to access the deployed application over the internet.

7. **Stopping the Deployment:**
    - When the deployment is no longer needed, the system allows the user to stop the Docker container, clean up the resources (including the cloned repository), and terminate the Ngrok process.

User Experience:

The system is designed for non-expert users, abstracting away the technical details of deployment. The only user input required is the GitHub repository URL, and the rest of the process is handled automatically. This makes the system ideal for developers and teams who want to quickly deploy and test applications without spending time on configuration or troubleshooting.

By leveraging Docker for application containerization and Ngrok for exposing the application to the public, the system provides a seamless and efficient deployment process.

3. METHODOLOGY

The methodology of the **Deployment Management System** is centered around automating the deployment of applications from GitHub repositories into Docker containers and exposing them to the public internet using Ngrok. The process is designed to be as seamless and straightforward as possible, requiring minimal input from the user while handling the complexities of containerization, image building, and public exposure. Below is a detailed explanation of the methodology, broken down into its main steps:

1. Cloning the Repository

The first step in the deployment process is cloning the GitHub repository, which contains the application’s source code, configuration files, and dependencies.

- **User Input**: The user provides the **GitHub repository URL**. This URL points to the source code repository of the application they wish to deploy.
- **Cloning Process**:
    - The system uses the **git Python module** to clone the repository from GitHub. The module interacts with Git to pull the application’s code from the remote repository.
    - The repository is stored in a predefined directory on the server, typically located in the ~/deployed_repos folder.
    - If the repository has already been cloned, the system will check for its existence and skip the cloning process, avoiding unnecessary duplication.
- **Repository Path**: The cloned repository is stored in a specific folder on the local server. This ensures the system can work with multiple deployments simultaneously and access each repository's files as needed.

2. Building the Docker Image

Once the repository is cloned, the system moves on to the next step: building a Docker image from the application’s source code.

- **Dockerfile Check**: The system first checks for the presence of a **Dockerfile** within the repository. A Dockerfile is essential as it contains instructions on how the application should be built and run in a Docker container. If no Dockerfile is found, the deployment process fails, and an error is raised.

- **Building the Image**:
  - If a valid Dockerfile exists, the system uses Docker's **Python SDK** (docker.from_env()) to build the Docker image from the repository's code.
  - The system runs the **docker.build()** method, which reads the Dockerfile and creates an image based on its instructions. The image is tagged with a unique identifier that includes the repository name, which helps differentiate images created from different repositories.
  - The Docker image contains all the necessary components to run the application, including the application's source code, dependencies, and any runtime configurations defined in the Dockerfile.
- **Image Validation**: If the image is successfully built, it is stored locally in the Docker environment. If any issues arise during the build process (e.g., missing dependencies, incorrect Dockerfile syntax), the system logs the error and halts the process.

3. Running the Docker Container

After building the Docker image, the system proceeds to run the application in a Docker container.

- **Container Initialization**:
  - The system runs the newly built Docker image as a **container** using the **docker.containers.run()** method from the Python SDK.
  - The container is configured to expose port **8080**, which is commonly used for web applications and APIs. This ensures the deployed application can be accessed via the specified port.
  - The system detaches the container to run in the background, meaning it operates independently of the terminal session, allowing for continuous operation without user interaction.

- **Logging**: The system logs the **container ID** for future reference. This ID is used to monitor and control the container during the deployment lifecycle. It allows the system to stop or remove the container when no longer needed.
- **Error Handling**: If there is an issue running the container (e.g., incorrect configuration, missing dependencies), the system captures the error and notifies the user, ensuring transparency in the deployment process.

4. Starting Ngrok

Once the container is running, the system starts **Ngrok**, a tool that creates a secure tunnel to expose the local application to the public internet.

- **Starting Ngrok**:
  - Ngrok is executed as a background process using the **subprocess** module, with a command to expose the local port (8080) to the internet. The system runs the command ngrok http 8080 to initiate this process.
- **Public URL Generation**:
  - After Ngrok is started, the system waits for a brief period (to ensure Ngrok has fully initialized). It then queries Ngrok's API by making a **GET** request to http://localhost:4040/api/tunnels to retrieve the list of active tunnels.
  - The response from Ngrok includes the public URL of the exposed tunnel. The system extracts the **HTTPS URL** from the response, which provides secure access to the application running in the Docker container.
- **User Access**: The generated public URL is provided to the user. This URL allows them to access the deployed application over the internet, regardless of their local machine's network configuration.

5. Deployment Process (Overall Workflow)

The entire deployment process is triggered by the function **deploy_repository()**, which automates the following steps:

1. **Cloning the Repository**: The system clones the repository from GitHub to a local directory.

2. **Building the Docker Image**: The system checks for a valid Dockerfile and builds the image.
3. **Running the Docker Container**: The image is run in a Docker container, exposing the application on port 8080.
4. **Starting Ngrok**: Ngrok is used to expose the application to the public internet, generating a secure URL.

Once the deployment is complete, the user receives a public URL through which they can access their application.

6. Stopping the Deployment

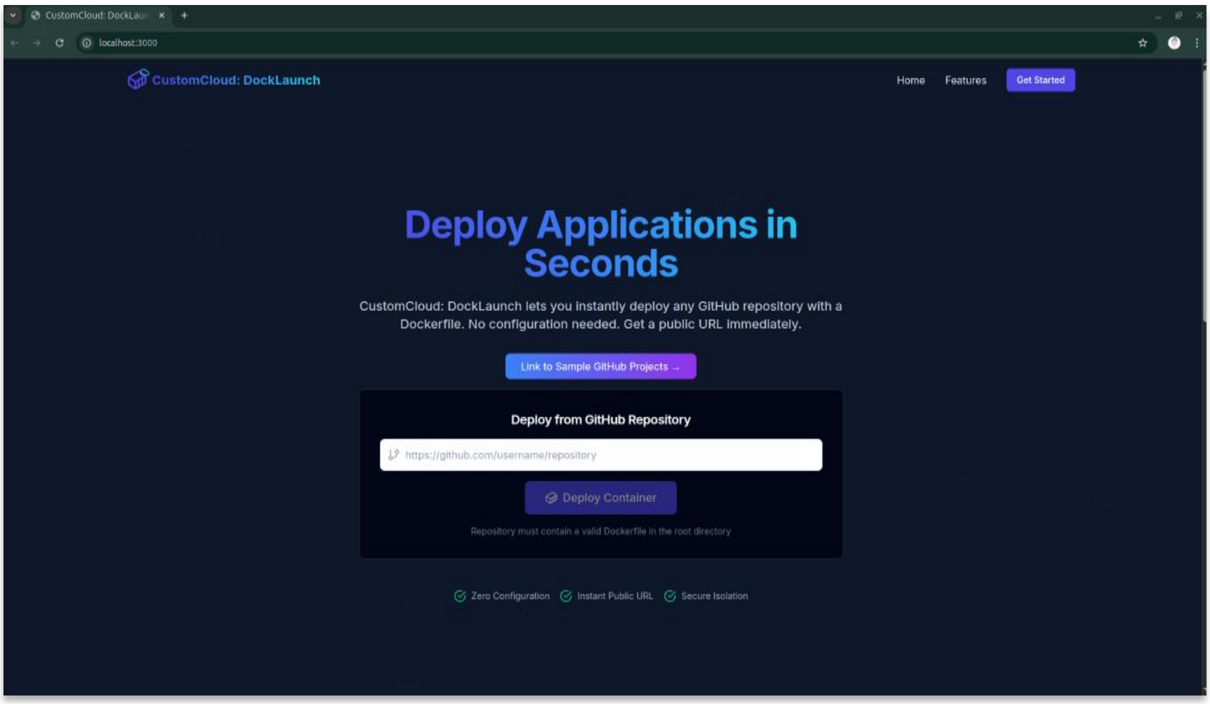
The system also includes functionality to stop the deployment and clean up resources.

- **Stopping the Container**: The system allows the user to stop the running Docker container. This is done by referencing the container ID, which was logged during the container run phase.
- **Removing the Container**: After stopping the container, the system removes it from the Docker environment to free up resources. This ensures that no unnecessary containers are left running.
- **Cleaning Up the Image**: The Docker image created during the deployment is also removed. This step ensures that no dangling or unused images occupy space in the system.
- **Deleting Cloned Repository**: The cloned repository is deleted from the local server after the deployment process ends. This prevents the accumulation of unnecessary data on the server.
- **Terminating Ngrok**: The Ngrok process is terminated to stop the tunnel and release any network resources used.

4. RESULTS

The deployment management system was successfully implemented and tested. The system's performance can be summarized as follows:

- Repository Cloning: The system successfully clones any valid GitHub repository, whether public or private (with appropriate access tokens). The process is efficient and error-free under normal conditions.
- Docker Image Building: The Docker image is built reliably from the cloned repository, provided the repository contains a valid Dockerfile. In cases where the Dockerfile is missing or incorrect, the system raises an appropriate error.
- Container Deployment: The Docker container runs smoothly with minimal resource consumption. The system ensures that the application inside the container is correctly exposed via port 8080.
- Ngrok Public URL: Ngrok generates a public URL successfully within a few seconds of starting the service. The public URL can be accessed from any device with an internet connection, providing a seamless way to interact with the deployed application.
- Error Handling: Proper error handling was implemented to ensure that issues such as missing files (e.g., Dockerfile), network problems, or Docker configuration errors are logged with clear messages for the user.



Deploy Applications in Seconds

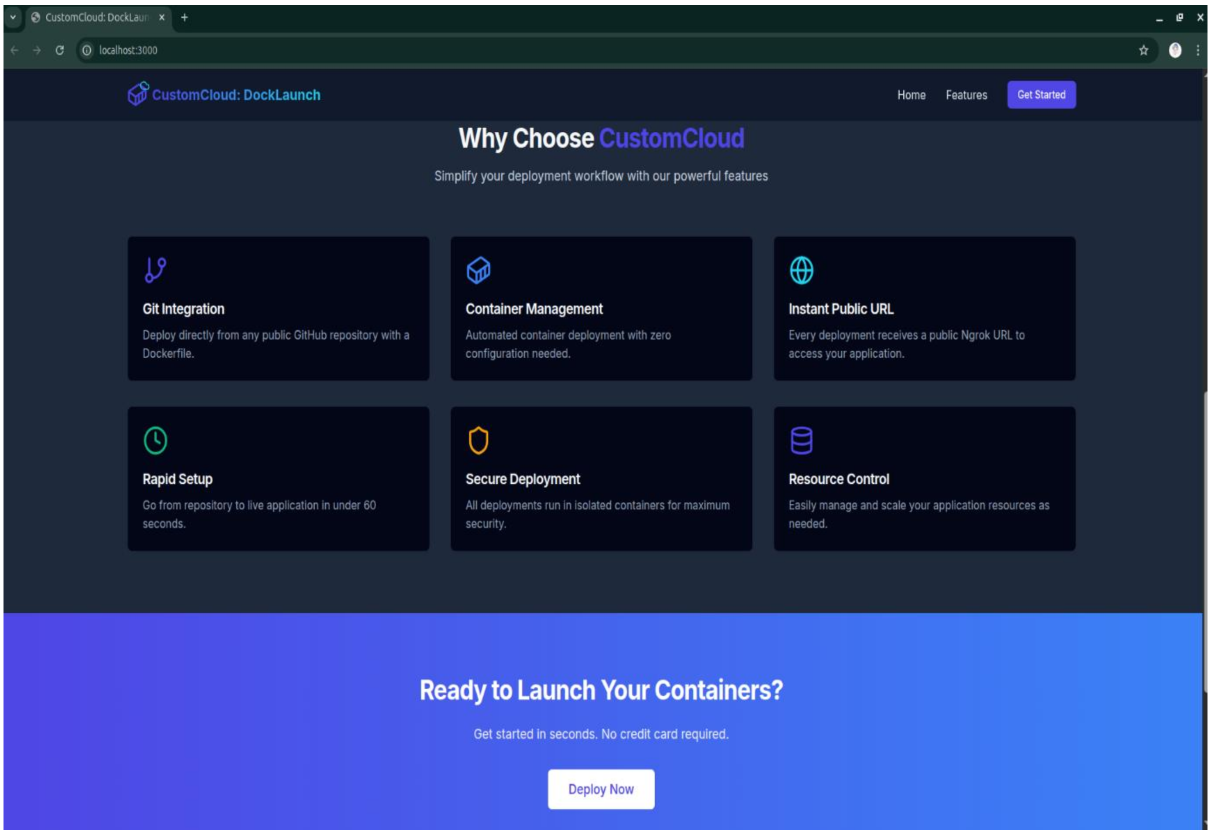
CustomCloud: DockLaunch lets you instantly deploy any GitHub repository with a Dockerfile. No configuration needed. Get a public URL immediately.

Deploy from GitHub Repository

https://github.com/username/repository

Repository must contain a valid Dockerfile in the root directory

Figure-1: Landing Page for DockLaunch


CustomCloud: DockLaunch

Home Features Get Started

Why Choose CustomCloud

Simplify your deployment workflow with our powerful features

Git Integration
Deploy directly from any public GitHub repository with a Dockerfile.

Container Management
Automated container deployment with zero configuration needed.

Instant Public URL
Every deployment receives a public Ngrok URL to access your application.

Rapid Setup
Go from repository to live application in under 60 seconds.

Secure Deployment
All deployments run in isolated containers for maximum security.

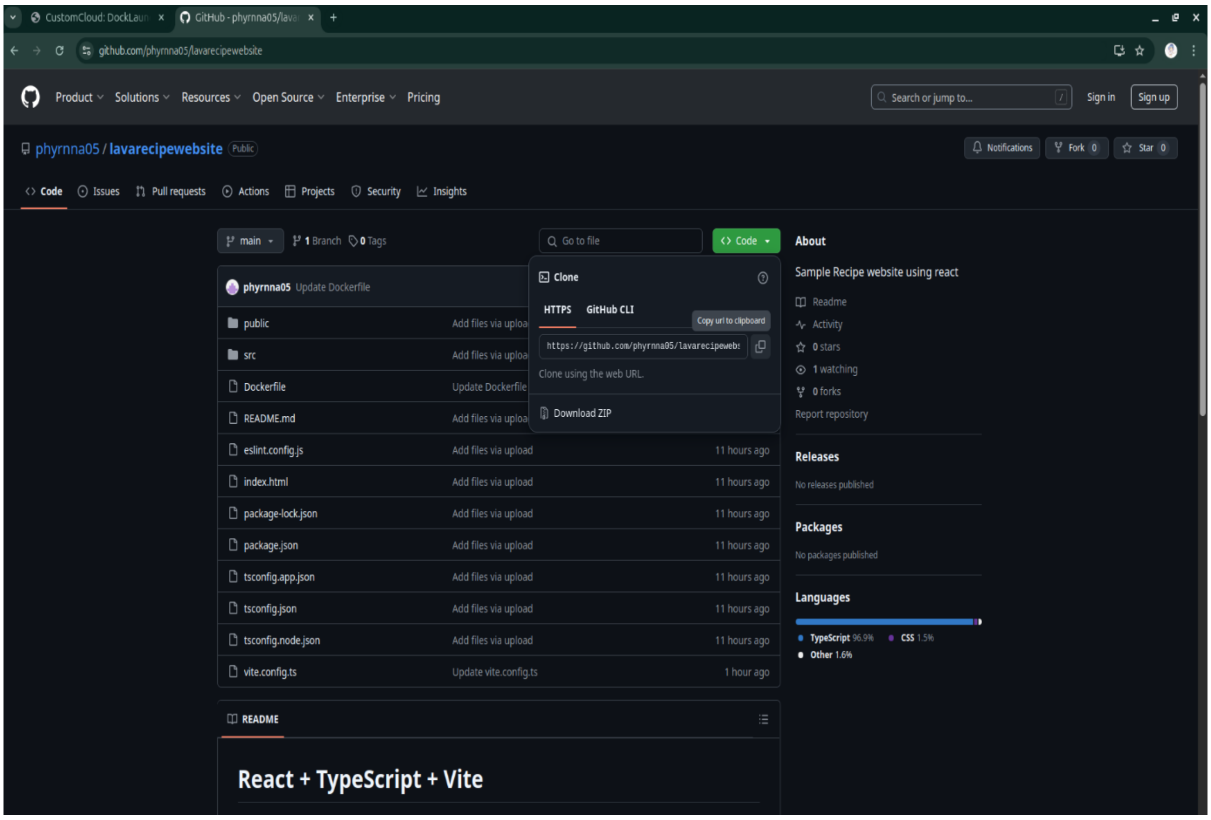
Resource Control
Easily manage and scale your application resources as needed.

Ready to Launch Your Containers?

Get started in seconds. No credit card required.

Deploy Now

Figure-2: Feature Section on Landing Page


github.com/phyrnna05/lavarecipewebsite

phyrnna05 / lavarecipewebsite Public

main · 1 Branch · 0 Tags

phyrnna05 Update Dockerfile

public	Add files via upload	
src	Add files via upload	
Dockerfile	Update Dockerfile	
README.md	Add files via upload	
eslint.config.js	Add files via upload	11 hours ago
index.html	Add files via upload	11 hours ago
package-lock.json	Add files via upload	11 hours ago
package.json	Add files via upload	11 hours ago
tsconfig.app.json	Add files via upload	11 hours ago
tsconfig.json	Add files via upload	11 hours ago
tsconfig.node.json	Add files via upload	11 hours ago
vite.config.ts	Update vite.config.ts	1 hour ago

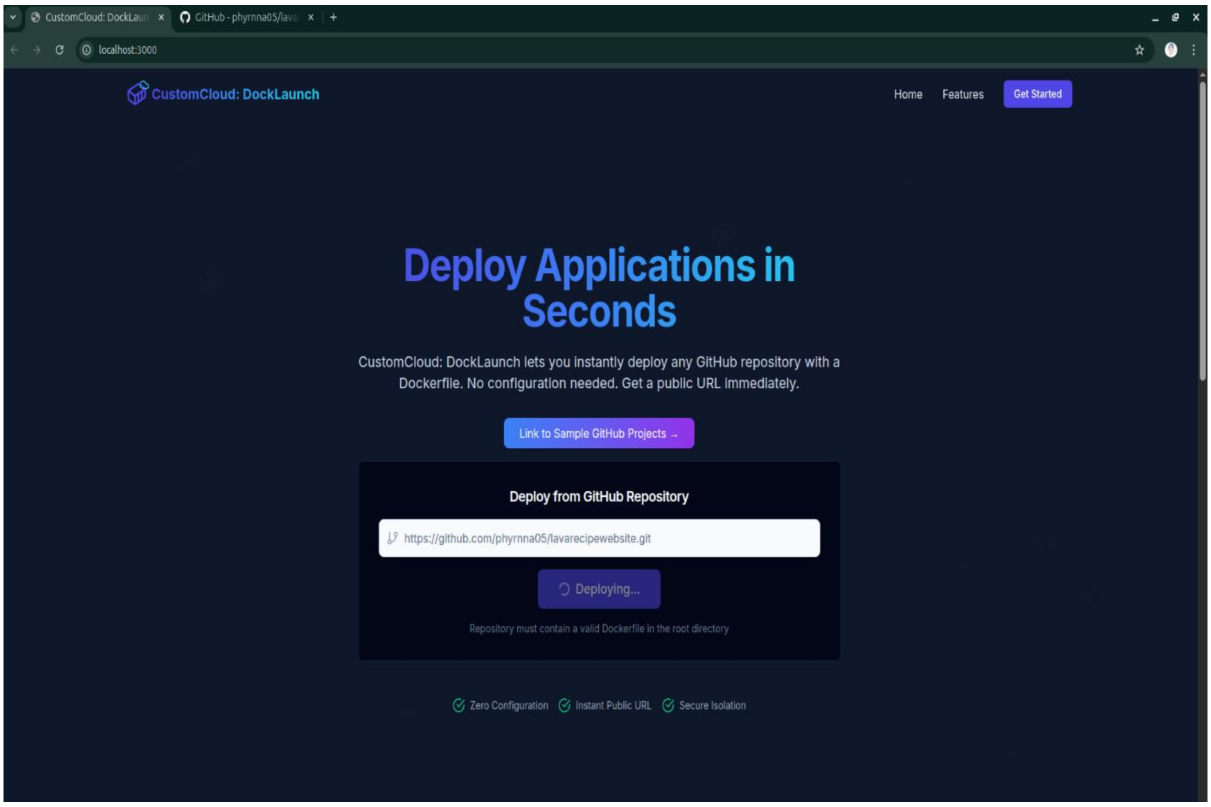
Clone — HTTPS: `https://github.com/phyrnna05/lavarecipewebs` — Clone using the web URL. Download ZIP

About: Sample Recipe website using react — 0 stars, 1 watching, 0 forks. Releases: No releases published. Packages: No packages published. Languages: TypeScript 96.9%, CSS 1.5%, Other 1.6%

README

React + TypeScript + Vite

Figure-3: Sample React Project Repository



Deploy Applications in Seconds

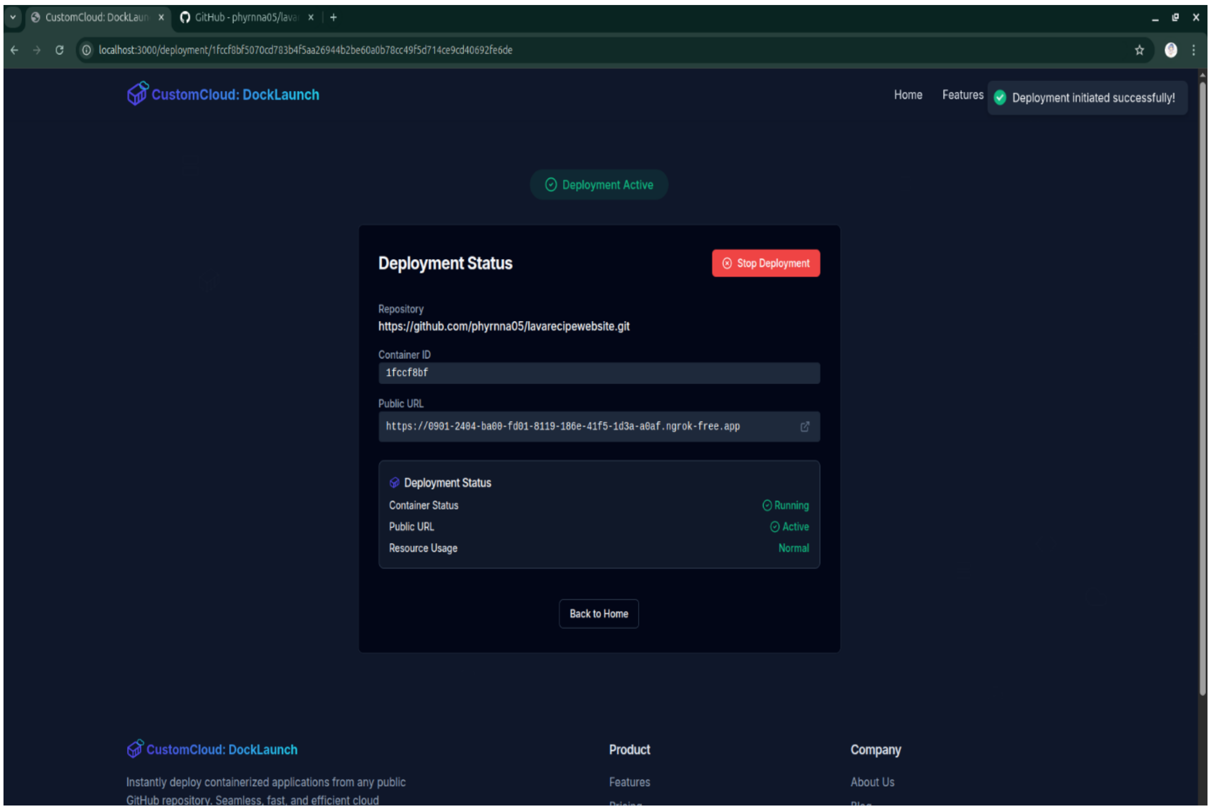
CustomCloud: DockLaunch lets you instantly deploy any GitHub repository with a Dockerfile. No configuration needed. Get a public URL immediately.

Deploy from GitHub Repository

https://github.com/phyrnna05/lavarecipewebsite.git

Repository must contain a valid Dockerfile in the root directory

Figure-4: Deploying a Application via Github repo link



Deployment initiated successfully!

Deployment Status

Repository
https://github.com/phyrnna05/lavarecipewebsite.git

Container ID
1fccf8bf

Public URL
https://0901-2404-ba00-fd01-8119-186e-41f5-1d3a-a0af.ngrok-free.app

Deployment Status

Container Status	Running
Public URL	Active
Resource Usage	Normal

Instantly deploy containerized applications from any public GitHub repository. Seamless, fast, and efficient cloud

Figure-5: Application Deployed Successfully with Active Status

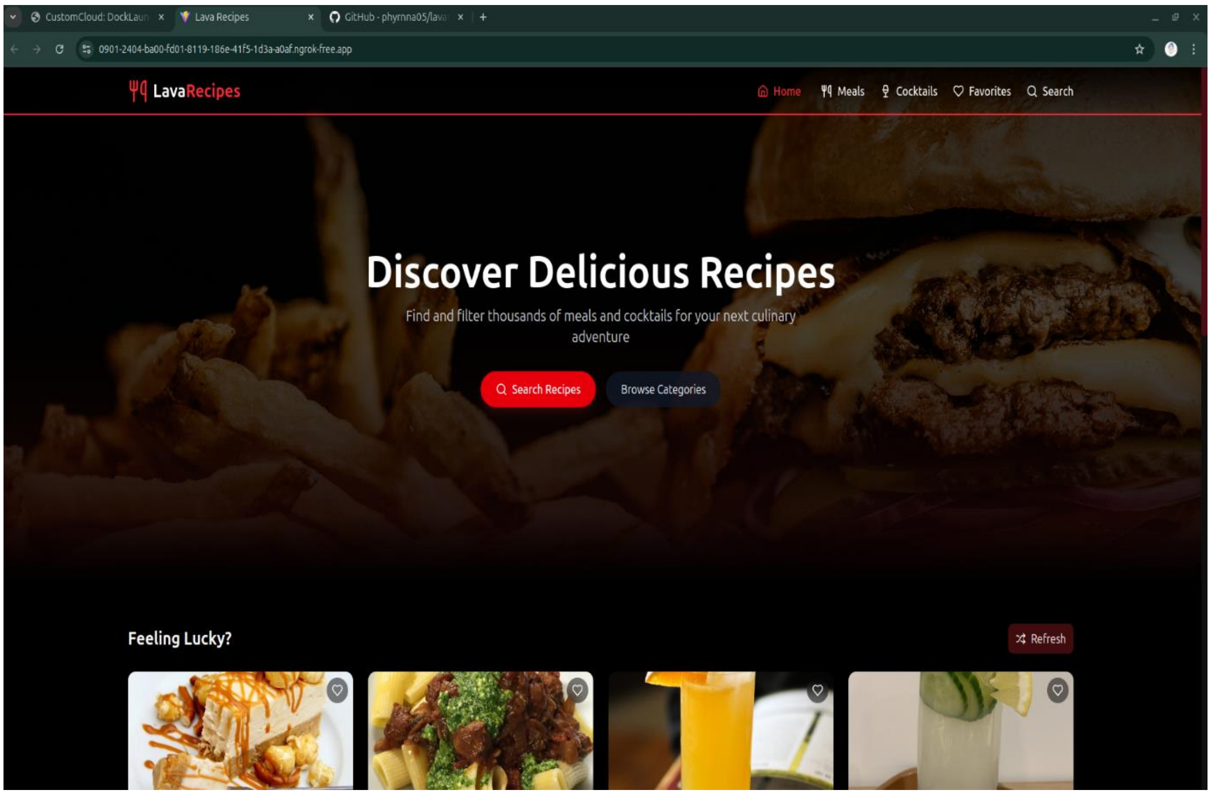

CustomCloud: DockLaun
Lava Recipes
GitHub - phyrnna05/lava
0901-2404-ba00-fd01-8119-186e-41f5-1d3a-a0af.ngrok-free.app
LavaRecipes
Home
Meals
Cocktails
Favorites
Search
Discover Delicious Recipes
Find and filter thousands of meals and cocktails for your next culinary adventure
Search Recipes
Browse Categories
Feeling Lucky?
Refresh

Figure-6: Hosted Application can now be accessed with Public url

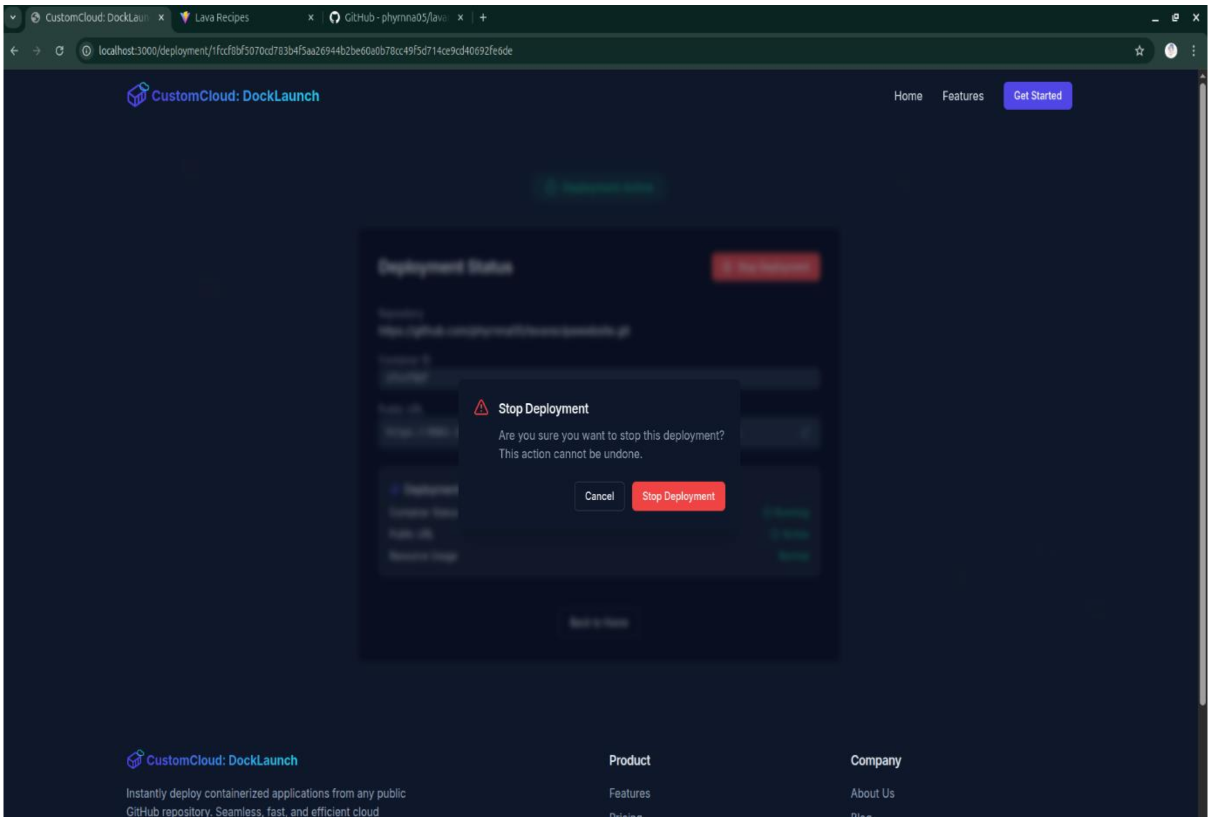

CustomCloud: DockLaun
Lava Recipes
GitHub - phyrnna05/lava
localhost:3000/deployment/1fccf8bf5070cd783b4f5aa26944b2be60a0b78cc49f5d714ce9cd40692fe6de
CustomCloud: DockLaunch
Home
Features
Get Started
Stop Deployment
Are you sure you want to stop this deployment? This action cannot be undone.
Cancel
Stop Deployment
CustomCloud: DockLaunch
Instantly deploy containerized applications from any public GitHub repository. Seamless, fast, and efficient cloud
Product
Features
Company
About Us

Figure-7: Stopping the Application

5. FUTURE SCOPE

While the current system provides a solid foundation for deployment, several improvements and features could be added to enhance its capabilities and scalability. Some potential future developments include:

- **Support for Multiple Repositories:**The system can be extended to handle the deployment of multiple repositories simultaneously. This would involve managing multiple Docker containers and Ngrok tunnels efficiently.
- **User Authentication and Permissions:** Implementing authentication mechanisms to ensure that only authorized users can deploy repositories. This could include GitHub OAuth integration for user authentication.
- **Scaling with Kubernetes:** For handling a larger number of users and deployments, integrating a container orchestration platform like Kubernetes would allow for better management of resources, auto-scaling, and fault tolerance.
- **Advanced Monitoring:** Adding real-time monitoring of the containers and applications. This could include CPU, memory, and network usage statistics, as well as logs from the applications running inside the containers.
- **Improved Security:** Enhancing security by isolating the deployed applications more strictly (e.g., using network policies in Docker), and ensuring that only necessary ports and services are exposed to the public.
- **Support for More Cloud Providers:** Expanding the system to support deployments to cloud platforms such as AWS, Azure, or GCP, enabling users to deploy their applications on scalable cloud infrastructure instead of local servers.
- **Error Handling and Debugging Tools:** Enhancing error detection by providing more detailed debugging information. This could include better handling of Docker and Ngrok errors, as well as automatic rollback on failed deployments.

6. CONCLUSION

The Deployment Management System developed in this project provides a streamlined and automated solution for deploying applications from GitHub repositories into Docker containers. By leveraging Docker for containerization and Ngrok for exposing applications to the public internet, the system simplifies the deployment process for developers and ensures that their applications can be tested and accessed with minimal configuration.

The project successfully demonstrates the power of Docker and Ngrok in building a deployment platform that can be easily extended and scaled in the future. The system offers a reliable way for developers to deploy and share their applications, and it sets the stage for further advancements in cloud-based deployment solutions.

7. REFERENCES

- https://docs.djangoproject.com/en/5.2/
- https://react.dev/
- https://docs.docker.com/