**Containerized URL Shortening Service: A DevOps Approach**

**Vanam Chandra Shekar**

**Redg no** : 12206976

School of Computer Science and Engineering, Lovely Professional University, Phagwara, Punjab, India

**Email**: [vanamchandrashekar@outlook.com](mailto:vanamchandrashekar@outlook.com)

**Abstract** : The project contains a containerized URL shortening service specifically focused on improving the efficiency and scaling of URL management. The service is built in Go using PostgreSQL as its database and has been containerized using Docker to make sure the service can be deployed consistently regardless of the environment. Docker Compose was used to manage the multi-container architecture while Jenkins can be used as an automated CI/CD pipeline to control the processes for building, testing and deploying. By integrating the DevOps processes, the shortener application is made to be more reliable and maintainable. Index Terms — URL Shortening, Containerization, DevOps, CI/CD, Docker, Jenkins

# I. Introduction

URL shortening services are an important need in the digital world. They allow easy sharing and tracking of links in a shorter and concise manner with lots of applications including social media and marketing campaigns. This project is to create a scalable URL shortening service with modern DevOps practices to help with deployment and manageability. This project will use containerization and CI/CD practices to address challenges such as consistent environment and quick updates. By including DevOps practices, the project will promote continuous updates, scalability, reliability, and improved resiliency in application infrastructure.

**II. Problem Statement and Motivation**

Traditional workflows to deploy applications can face obstacles with scalability, consistency and maintenance. This is particularly the case in environments where working with manual deploy and updates can introduce errors, as well as inefficiency. Additionally, the work processes often have inconsistencies between different environments that result in variability in where the application is running

The motivation for this project is to tackle the challenges of traditional workflows by using containers and automation. By adopting Docker, we can package the application to run consistently across different environments, eliminating discrepancies. Integrating CI/CD with Jenkins automates the build, test, and deployment processes, reducing human error and speeding up development. This approach enhances reliability, allowing for more frequent and seamless deployments. The project aims to demonstrate how Dockerized applications, deployed through CI/CD, can achieve better, faster, and more efficient deployment**.**

**III. System Architecture**

The system is designed to be modular, ensuring that each component integrates smoothly and operates efficiently. This approach allows for flexibility and scalability, making it easier to manage and expand the service as needed.

**Core Application**

* Written in Go, where URL shortening logic and request handling happen.
* Connects with the PostgreSQL database for persisting URL mappings.

**Containerization:**

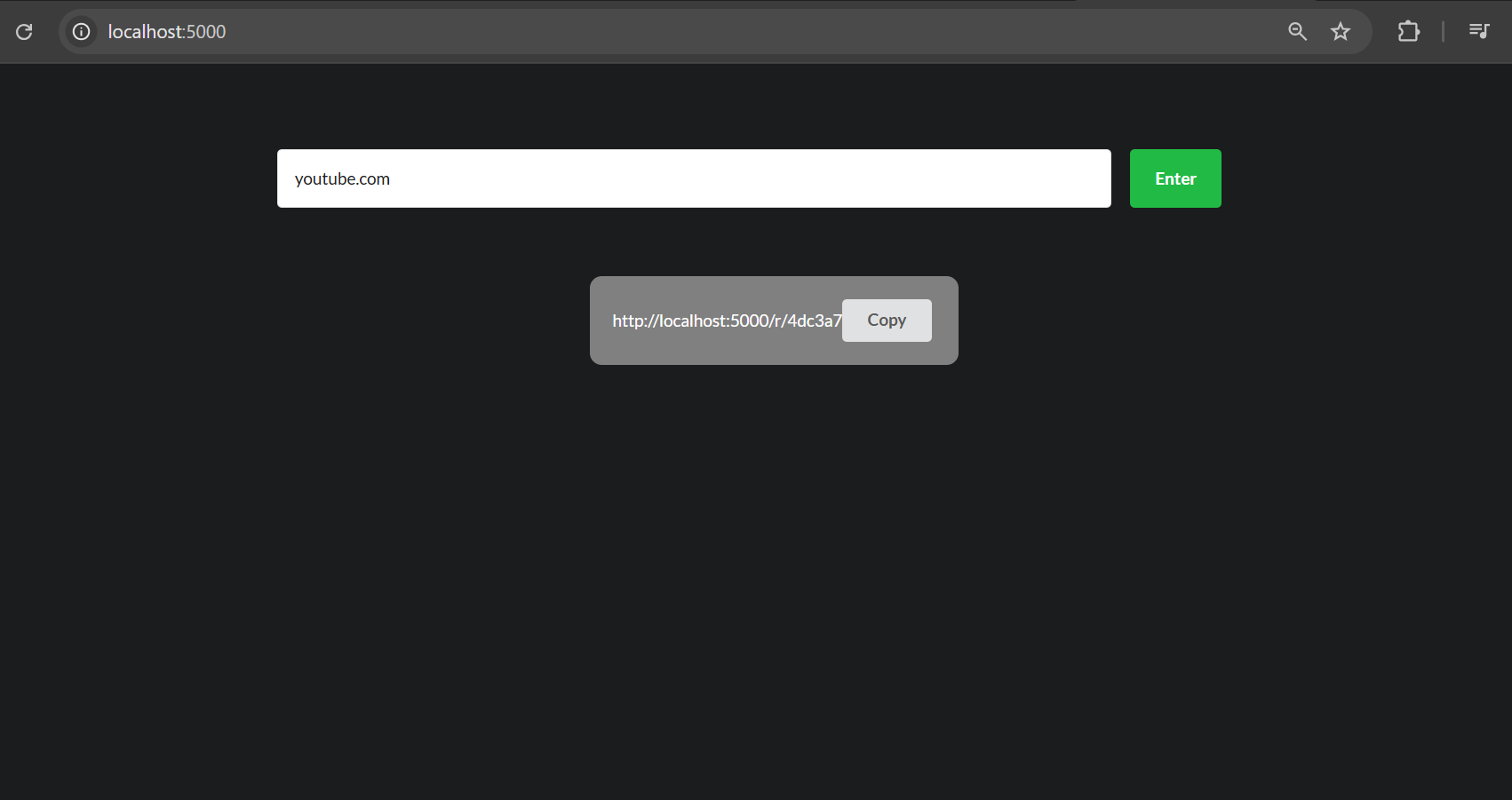
* Utilizes Docker for consistent application behavior across multiple environments.
* Application and database are packaged in containers for isolation and version control.

**CI/CD Pipeline:**

* Used Jenkins for build-testing-deploy automation.
* Quick and reliable deployment of the application, interacting with Docker containers.

**Advantages:**

* Modular design allows the system to change and scale according to need.

****

**IV. Data Management**

Data management is critical to the operation and trustworthiness of the URL shortening service. In this section, we describe how the application structures, stores, and accesses data.

**Database Design:**

* The application uses PostgreSQL to store URL mappings.
* The schema consists of tables that store short codes, original URLs, and timestamp of record creation.

**Data Interactions:**

* The Go application interacts with the database through SQL.
* The application uses functions to insert new URLs and retrieve the original URL from a shorted code.

**Data Integrity:**

* Unique short codes are guaranteed by database constraints.
* The database is backed up regularly to mitigate data loss.

**Scalability:**

* The database design assists in scaling to accommodate increased data.
* Queries are optimized to ensure performance is maintained should the dataset grow.

**V. Application Development**

The application development process consists of a user-friendly interface and a competent backend to make URL shortening and redirection as efficient as possible.

**User-interface (UI):**

* This was constructed out of HTML and uses Go templates for dynamic rendering.
* The presentation allows for a clean form in which users enter their URLs and get a shortened link.
* Used for warnings, errors and feedback on how to proceed with all outcomes.

**Backend Logic:**

* The backend processing is implemented in Go specifically for URL shortening, storing and redirecting.
* Middleware to log each process, recover from any error states, and implement rate limitation to improve performance and establish basic security.
* Uses hashing algorithms to make links from short codes to the actual URL.

**Testing and Validating:**

* Performs tests to highlight edge cases and errors that the application will experience in use, in order to find out the best errors to report back to users.
* Validates that users' inputs are not invalid URLs so they skip processing stage.
* Unit tests of functions that are considered most important to maintain development quality.

**Scalability:**

* Built to work with more traffic and data as they grow with an increased user-based level.
* As with coding practices, all code was optimized to have fewer indistinguishable database queries under load.

**VI. Container and Containerization via Docker**

Containerization is important in this project by ensuring everything works and behaves the same across environments and ultimately speeds up deployments.

**Dockerized:**

* The application is collected and compiled in a docker container encapsulating all the dependencies and compilation settings.
* The repository contains a multi-stage Dockerfile which eliminates unnecessary files and configurations and ultimately brings the final image size down.

**Consistency:**

* Docker provides a consistent execution environment when working in development, testing and production environments.
* Containers also isolate the software from that of the host, removing overwrites or conflicts with other software.

**Port Structure:**

* The application and database are setup to expose ports specific to their requirements, also managing ports that allow for internal communication between containers possible, as well as access to the exposed ports from outside the docker network.
* Docker compose is used to create and configure across multiple services.

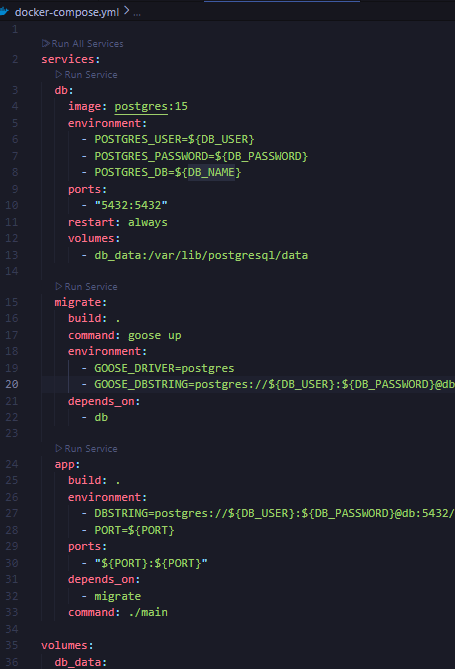
**Advantages of containerization:**

* This forms a consistent execution environment for deployments.
* Easier to scale since, technologies can be easily copied and distributed across multiple nodes.

**Security:**

The permissions give the application only the minimal rights to execute, these settings is coupled with environmental policies and procedures to secure configuration, and ensures the monitoring tools are updated and do routinely vulnerability scans.





**VII. CI/CD Pipeline with Jenkins**

The CI/CD pipeline is an integral part of the project, allowing for the build, testing, and deployment of applications to happen quickly and reliably by automating those processes.

**Pipeline Setup:**

* Jenkins is used for automating the continuous integration and continuous deployment process.
* The Jenkinsfile that defines the pipeline has the setup, build, test, and deployment stages.

**Build Automation:**

* The pipeline automatically builds Docker image meant to run the application so the developer doesn’t have to build, pull, or push the application images manually.
* Utilizes Docker to manage builds, adds, and updates to the application’s Docker image.

**Testing and Quality Assurance:**

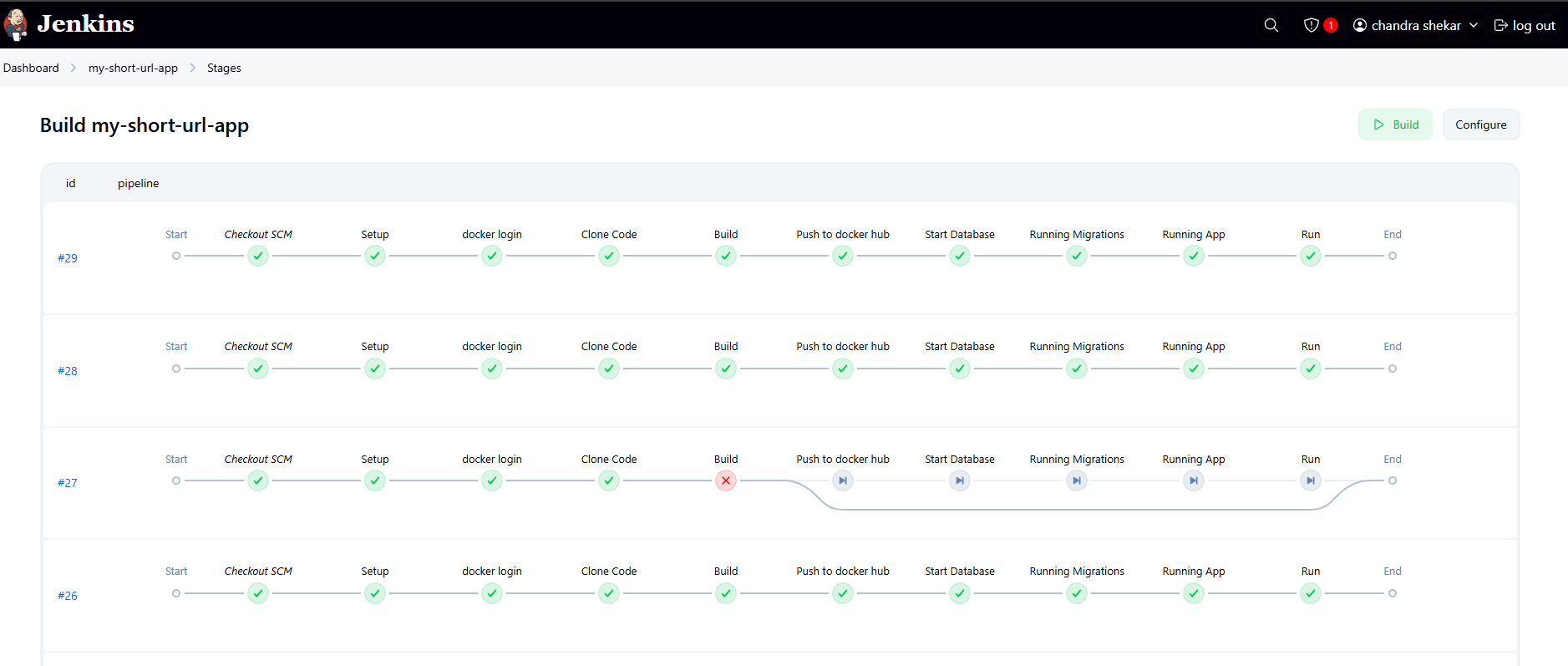
* There are tests that are automated as part of the pipeline using Spotless that allows our team to catch errors at early stages of development.
* Codebases don’t go into production until all tests have passed.

**Deployment Process:**

* The deployment is done to the production environment using Docker Compose.
* The pipeline also offers the means to act as a base for automating starting and orchestrating the containers with Docker.

**Error Handling and Notifications:**

* The pipeline does include error handling to accommodate build failures or deployment issues.
* A notification system was designed to inform the development team of any issues and to facilitate a rapid repair.



**VIII. Deployment Strategy**

**Methods:**

**Manual Deploymen**t: Involves manually building Docker images and running containers using Docker commands. This method is useful for initial testing and small-scale deployments.

**Automated Deployment:** Utilizes Jenkins to automate the deployment process, ensuring consistency and reducing manual errors. The pipeline is triggered by changes in the codebase.

**GitHub Hooks:**

GitHub webhooks are configured to trigger the Jenkins pipeline automatically whenever changes are pushed to the repository.

This integration ensures that the latest code is always built, tested, and deployed without manual intervention, streamlining the development workflow.

**Docker Hub Integration:**

Docker images are pushed to Docker Hub, allowing for easy distribution and version control.

Automated builds and updates are managed through the CI/CD pipeline, ensuring the latest version is always available.

**IX. Issues and Solutions**

**Technical Problems:**

* Difference in environments between development and production
* Dependencies and systems compatibility

**Solutions:**

* Docker was implemented to standardize environments and minimize differences
* A multi-stage Dockerfile was used to more effectively manage dependencies and reduce image size.

**X. Future Enhancements**

**Scaling with Kubernetes:**

* Leverage Kubernetes for container orchestration to allow for dynamic scaling and advanced resource management.

**Adding Authentication:**

* Add user authentication to secure the application, which could be based on OAuth, or by using existing identity providers.

**Implementing Monitoring Tools:**

* Implement monitoring tools such as prometheus and grafana for real time performance monitoring and logging.

**User Experience Improvements:**

* Add features like analytics dashboards, and customized short URL support to further engage users.

**Considering Serverless Architectures:**

* Begin to research and consider Serverless computing, to manage infrastructure, and increase scaling and costs.

**XI. Concluding Thoughts**

**Summary**: The project has demonstrated how DevOps practices can be integrated into an application development process, resulting in a URL shortening service that is both scalable and reliable.

**Importance**: The project also showcases, using containerization and CI/CD, the benefits of using modern deployment strategies to improve efficiency, consistency, and scalability.

**XII. References**

**Official Documentation:**

1. **Go**: Refer to the official Go documentation for language features, libraries, and best practices.

* URL: <https://golang.org/doc/>

1. **Docker**: Utilize Docker's official documentation for containerization guidelines, Dockerfile syntax, and best practices for image management.

* URL: <https://docs.docker.com/>

1. **Jenkins**: Consult Jenkins documentation for setting up CI/CD pipelines, configuring Jenkinsfiles, and integrating with other tools.

* URL: <https://www.jenkins.io/doc/>

**Additional Resources:**

1. **PostgreSQL**: Use PostgreSQL documentation for database setup, SQL queries, and performance tuning.

* URL: <https://www.postgresql.org/docs/>

1. **GitHub**: Explore GitHub's resources on webhooks and repository management to understand integration with CI/CD pipelines.

* URL: <https://docs.github.com/en>

**Project Links**

**GitHub** : <https://github.com/vanam-chandra-shekar/shortUrl.git>

**DockerHub**:<https://hub.docker.com/u/vanamchandra>