# CLOUD COMPUTING TECHNOLOGY PRACTICUM REPORT

**DEPLOYING BE TO CLOUD RUN AND RUNNING FE VIA DOCKER CONTAINER**

**PLUG - H**

By:

| | |
|---|---|
| **Nama** | **: Insyuzuu Cahyani 'Aisyah** |
| **NIM** | **: 123220013** |

**INFORMATICS STUDY PROGRAM INFORMATICS ENGINEERING DEPARTMENT INDUSTRIAL ENGINEERING FACULTY UNIVERSITAS PEMBANGUNAN NASIONAL "VETERAN" YOGYAKARTA**

**2025**

# LEGALIZATION PAGE

# FINAL REPORT

Arranged by :

Insyuzuu Cahyani ‘Aisyah      123220013

Checked and Approved by Practicum Assistant …… ……

At the date of: …………………

.

**Approve,.**

| | |
|---|---|
| **Practicum Assistant** | **Practicum Assistant** |
| **Muhammad Rafli** | **Sayang Sani** |
| **NIM 123210078** | **NIM 123210044** |

# PREFACE

First and foremost, I express my sincere gratitude to God Almighty for granting me the strength, knowledge, and perseverance to complete this final report on **Cloud Computing Technology**. This report presents an overview of cloud computing concepts, architectures, deployment models, and real-world applications, highlighting its benefits and challenges in modern industries.

I extend my heartfelt appreciation to my instructors and mentors for their guidance and support throughout this process. Their insights have been invaluable in shaping this report. Additionally, I acknowledge the contributions of researchers and industry experts whose work has been a vital reference.

I hope this report serves as a useful resource for those interested in cloud computing. While I have made every effort to ensure accuracy, I welcome any constructive feedback for future improvements.

Yogyakarta, March 18 2025

Insyuzuu Cahyani ‘Aisyah

# TABLE OF CONTENTS

# TABLE OF FIGURE

# CHAPTER I
# INTRODUCTION

## 1.1 Background

In today's digital era, the need for fast, secure, and easily accessible information recording is increasing. A note-taking application, or notes app, has become one of the popular solutions used by a wide range of users, including students, academics, and professionals. This type of application allows users to store important notes online, manage task lists, and organize creative ideas efficiently.

To understand the implementation of modern technologies in web application development, a notes app project has been created using a client-server architecture. The application is built using Express.js for the back-end, Sequelize as the ORM for managing the MySQL database, and HTML/JavaScript for the front-end. To support scalability and service availability, the back-end is deployed using Cloud Run from Google Cloud Platform, while the front-end is run via a Docker container on Google Cloud.

This assignment not only aims to develop a functional application but also to provide practical experience in the processes of containerization, cloud deployment, and the separation of front-end and back-end architecture in a modern web application.

## 1.2 Problem Formulation

1. How to build a notes application using a client-server architecture?
2. How to effectively deploy the back-end to Cloud Run using Docker containers?
3. How to run the front-end via a Docker container in Google Cloud?
4. How to apply Docker-based containerization and cloud deployment in a real web project?

## 1.3 Objective

The objective of this research:

1. To develop a functional notes application using a client-server architecture.
2. To containerize the back-end using Docker and deploy it to Cloud Run for scalability and accessibility.
3. To containerize the front-end using Docker and run it via Google Cloud.
4. To demonstrate the practical implementation of Docker-based containerization and cloud deployment in a modern web application project.

## 1.4 Benefits

This project provides practical experience in building web applications using a client-server architecture, allowing developers to understand how modern web systems are structured. It also offers hands-on learning of Docker for containerization, enabling efficient packaging and deployment of applications. Through the use of Cloud Run and Google Cloud, participants are exposed to real-world cloud deployment processes. The application is designed to be scalable and accessible, reflecting the structure of modern online services. Additionally, the project promotes a modular architecture by clearly separating the front-end and back-end, which supports better maintainability and scalability.

# CHAPTER II
# LITERATUR REVIEW

## 2.1 Modern Web Application Architecture

Web applications today are commonly built using a client-server architecture, which separates the front-end and back-end for better scalability, maintainability, and performance. Adopting a client-server model enhances modular development, as front-end and back-end components are developed and maintained independently, allowing for faster updates and scalability. This approach is increasingly popular in building applications that need to support large user bases, as it enables efficient communication between the client (user interface) and the server (database, business logic). By leveraging technologies like RESTful APIs or GraphQL, the communication between the two parts is streamlined, further improving the performance and flexibility of the application.

## 2.2 Containerization with Docker

Containerization has revolutionized how applications are deployed and managed, making it easier to run software across different environments. Docker, a popular containerization tool, allows developers to package applications along with all their dependencies into lightweight, portable containers. Docker containers ensure consistency across development, testing, and production environments, which is crucial for maintaining application stability. Docker also improves scalability, as containers can be deployed quickly and replicated as needed, providing flexibility to manage traffic spikes. By isolating the application’s environment, Docker makes it possible to avoid issues that typically arise from different operating systems or configurations.

## 2.3 Cloud Deployment with Cloud Run

Cloud computing has become a key enabler for modern web applications, offering scalability, reliability, and reduced infrastructure management overhead. Google Cloud's Cloud Run provides a serverless environment that automatically scales applications based on demand. Serverless architectures like Cloud Run offer significant advantages in terms of reduced costs, as users only pay for the compute time used. Cloud Run, in particular, allows developers to deploy Docker containers seamlessly, enabling easy integration of containerized applications with cloud services. By offloading the infrastructure management to the cloud provider, developers can focus more on building and improving their applications rather than worrying about server management.

# CHAPTER III
# METODOLOGI

## 3.1 Problem Analysis

The main goal of the project is to create a notes application with a client-server architecture, deploy the back-end on Cloud Run, and run the front-end via a Docker container on Google Cloud. The following are the key problems that arise during this process:

1. Separation of Front-End and Back-End:
   A clear distinction between the front-end (user interface) and back-end (server logic and database management) is necessary to ensure efficient development, scalability, and maintenance. This separation can introduce challenges in ensuring smooth communication between the two components, especially when integrating with cloud services.
2. Cloud Deployment with Scalability:
   Deploying the back-end to Cloud Run requires a scalable and reliable solution. Cloud Run automatically scales services, but ensuring the application is designed to handle traffic fluctuations without performance degradation can be complex.
3. Containerization of the Application:
   The front-end and back-end must be containerized using Docker to ensure portability and consistency across different environments. However, Dockerizing both components and managing them in separate containers poses challenges in keeping the application architecture simple while ensuring seamless interaction.
4. Environment Setup:
   Running the front-end from Google Cloud while ensuring interaction with the back-end deployed on Cloud Run requires a stable, containerized environment and proper configuration to ensure that both parts of the application communicate effectively.

## 3.2 Solution Design

To address the identified problems, the solution design incorporates several key elements. The solution adopts a client-server architecture, where the front-end is responsible for handling user interaction and the back-end is responsible for data processing and business logic. The front-end will be a web application built with HTML/JavaScript, while the back-end will be implemented using Express.js and connected to a MySQL database through Sequelize. To ensure consistency across different environments, both the front-end and back-end will be Dockerized. A Dockerfile will be created for the back-end to specify how to build and run the Express.js server, and the front-end will be packaged in a separate Docker container. This approach allows for the easy deployment and management of each component independently.

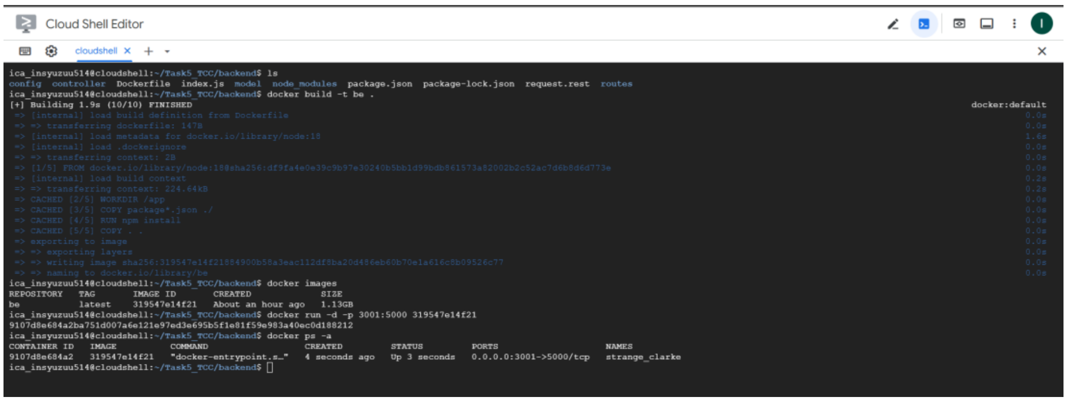
The back-end will be deployed on Google Cloud Run, a serverless platform that automatically scales the application based on demand, ensuring the service can handle varying amounts of traffic. The back-end will be containerized using Docker and deployed directly to Cloud Run, where it will be accessible through a REST API that the front-end can interact with. The front-end will run in a Docker container on Google Cloud, eliminating the need to deploy it to a cloud platform like Cloud Run. Google Cloud provides the necessary infrastructure for running containers, allowing for easy testing and execution of the application while maintaining a clear separation between the front-end and back-end. The front-end will interact with the back-end API hosted on Cloud Run via HTTP requests.

To ensure smooth communication between the front-end and back-end, the front-end will make API requests to the back-end hosted on Cloud Run, with the interaction being configured using environment variables. This allows the front-end to dynamically fetch the correct API endpoints based on the deployment configuration. Leveraging Google Cloud Run ensures the back-end scales automatically based on user demand, guaranteeing high availability and performance. The modular design of the application—by separating the front-end and back-end into distinct containers—ensures easier maintenance, updates, and future enhancements without disrupting the overall system.

# CHAPTER IV
# RESULT AND DISCUSSION

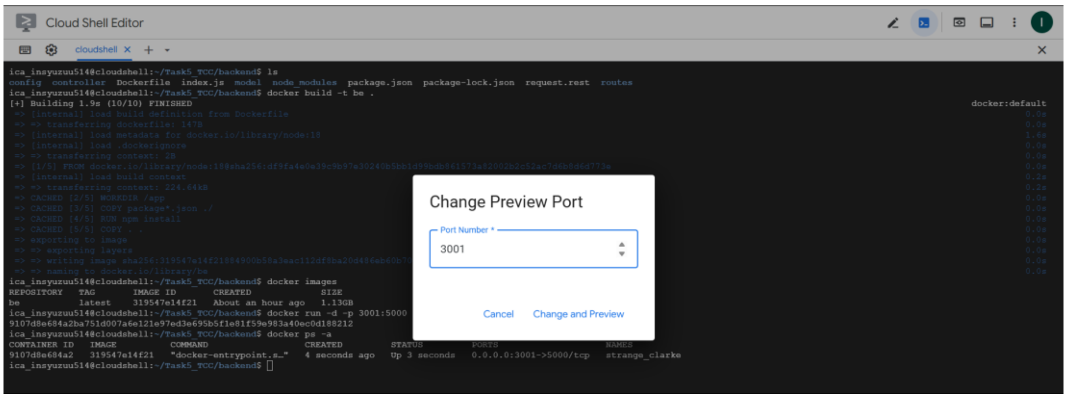
## 4.1 Result

The project successfully implemented a notes application using a client-server architecture, with the back-end deployed on Google Cloud Run and the front-end containerized and run via Google Cloud. The system demonstrated scalability, as the back-end automatically adjusted to traffic demands through Cloud Run. The communication between the front-end and back-end was effectively managed through API calls, ensuring smooth interaction. Docker containerization ensured the portability and consistency of the application across different environments. With the several step.



```
ica_insyuzuu514@cloudshell:~/Task5_TCC/backend$ ls
config  controller  Dockerfile  index.js  model  node_modules  package.json  package-lock.json  request.rest  routes
ica_insyuzuu514@cloudshell:~/Task5_TCC/backend$ docker build -t be .
[+] Building 1.9s (10/10) FINISHED                                                                      docker:default
 => [internal] load build definition from Dockerfile                                                              0.0s
 => => transferring dockerfile: 147B                                                                              0.0s
 => [internal] load metadata for docker.io/library/node:18                                                        1.6s
 => [internal] load .dockerignore                                                                                 0.0s
 => => transferring context: 2B                                                                                   0.0s
 => [1/5] FROM docker.io/library/node:18@sha256:df9fa4e0e39c9b97e30240b5bb1d99bdb861573a82002b2c52ac7d6b8d6d773e  0.0s
 => [internal] load build context                                                                                 0.2s
 => => transferring context: 224.64kB                                                                             0.2s
 => CACHED [2/5] WORKDIR /app                                                                                     0.0s
 => CACHED [3/5] COPY package*.json ./                                                                            0.0s
 => CACHED [4/5] RUN npm install                                                                                  0.0s
 => CACHED [5/5] COPY . .                                                                                         0.0s
 => exporting to image                                                                                            0.0s
 => => exporting layers                                                                                           0.0s
 => => writing image sha256:319547e14f21884900b58a3eac112df8ba20d486eb60b70e1a616c8b09526c77                      0.0s
 => => naming to docker.io/library/be                                                                             0.0s
ica_insyuzuu514@cloudshell:~/Task5_TCC/backend$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED             SIZE
be           latest    319547e14f21   About an hour ago   1.13GB
ica_insyuzuu514@cloudshell:~/Task5_TCC/backend$ docker run -d -p 3001:5000 319547e14f21
9107d8e684a2ba751d007a6e121e97ed3e695b5f1e81f59e983a40ec0d188212
ica_insyuzuu514@cloudshell:~/Task5_TCC/backend$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS                    NAMES
9107d8e684a2   319547e14f21   "docker-entrypoint.s…"   4 seconds ago   Up 3 seconds   0.0.0.0:3001->5000/tcp   strange_clarke
ica_insyuzuu514@cloudshell:~/Task5_TCC/backend$ 
```

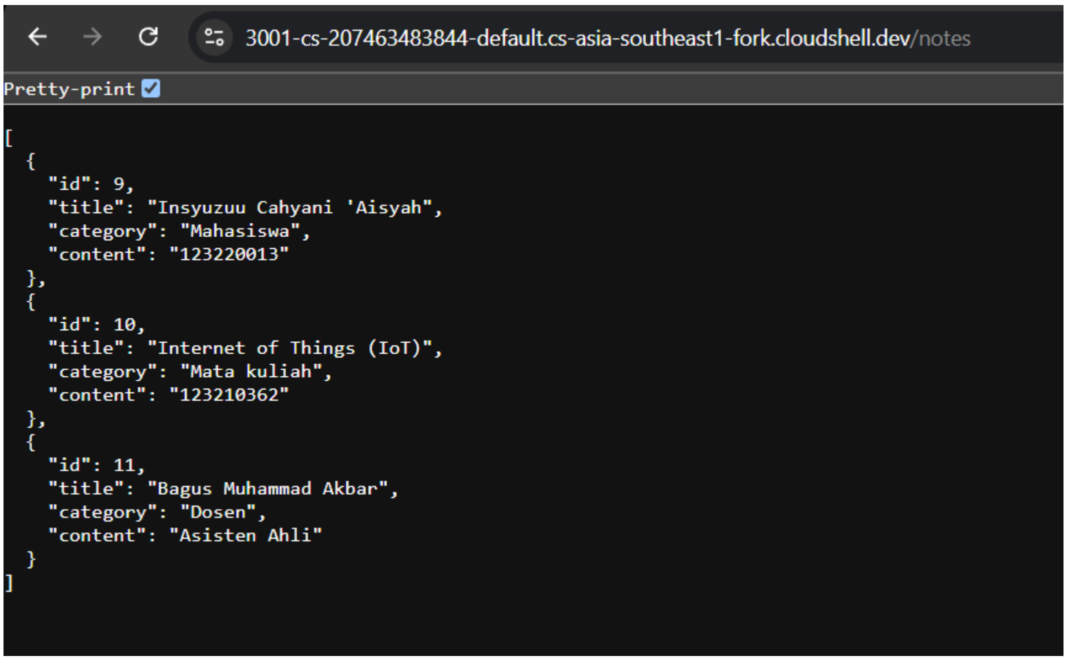
*Figure 4.1.1 Build the Docker Image*



Change Preview Port

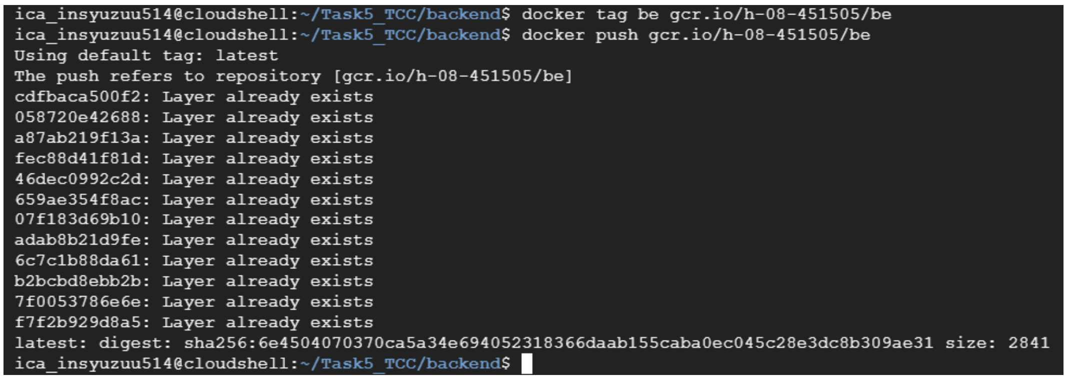
Port Number *
3001

*Figure 4.1.2 Change port*


3001-cs-207463483844-default.cs-asia-southeast1-fork.cloudshell.dev/notes
Pretty-print ✓

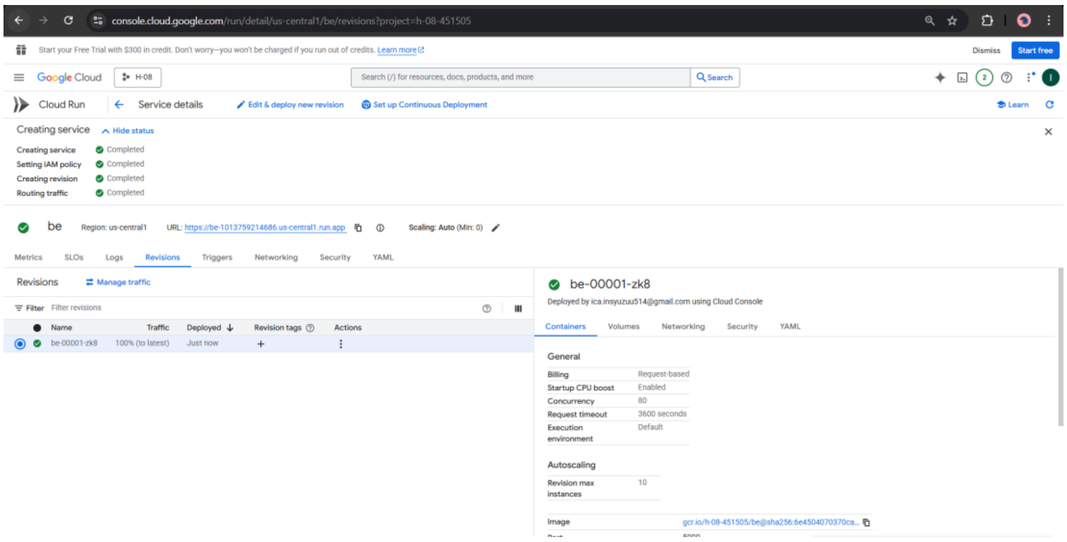
```
[
  {
    "id": 9,
    "title": "Insyuzuu Cahyani 'Aisyah",
    "category": "Mahasiswa",
    "content": "123220013"
  },
  {
    "id": 10,
    "title": "Internet of Things (IoT)",
    "category": "Mata kuliah",
    "content": "123210362"
  },
  {
    "id": 11,
    "title": "Bagus Muhammad Akbar",
    "category": "Dosen",
    "content": "Asisten Ahli"
  }
]
```

*Figure 4.1.3 Data for back-end*



```
ica_insyuzuu514@cloudshell:~/Task5_TCC/backend$ docker tag be gcr.io/h-08-451505/be
ica_insyuzuu514@cloudshell:~/Task5_TCC/backend$ docker push gcr.io/h-08-451505/be
Using default tag: latest
The push refers to repository [gcr.io/h-08-451505/be]
cdfbaca500f2: Layer already exists
058720e42688: Layer already exists
a87ab219f13a: Layer already exists
fec88d41f81d: Layer already exists
46dec0992c2d: Layer already exists
659ae354f8ac: Layer already exists
07f183d69b10: Layer already exists
adab8b21d9fe: Layer already exists
6c7c1b88da61: Layer already exists
b2bcbd8ebb2b: Layer already exists
7f0053786e6e: Layer already exists
f7f2b929d8a5: Layer already exists
latest: digest: sha256:6e4504070370ca5a34e694052318366daab155caba0ec045c28e3dc8b309ae31 size: 2841
ica_insyuzuu514@cloudshell:~/Task5_TCC/backend$
```

*Figure 4.1.4 Push docker to gcr*



console.cloud.google.com/run/detail/us-central1/be/revisions?project=h-08-451505

Start your Free Trial with $300 in credit. Don't worry—you won't be charged if you run out of credits. Learn more

Creating service

| | |
|---|---|
| Creating service | Completed |
| Setting IAM policy | Completed |
| Creating revision | Completed |
| Routing traffic | Completed |

be Region: us-central1 URL: https://be-1013759214686.us-central1.run.app Scaling: Auto (Min: 0)

| Name | Traffic | Deployed | Revision tags | Actions |
|---|---|---|---|---|
| be-00001-zk8 | 100% (to latest) | Just now | | |

be-00001-zk8

Deployed by ica.insyuzuu514@gmail.com using Cloud Console

General

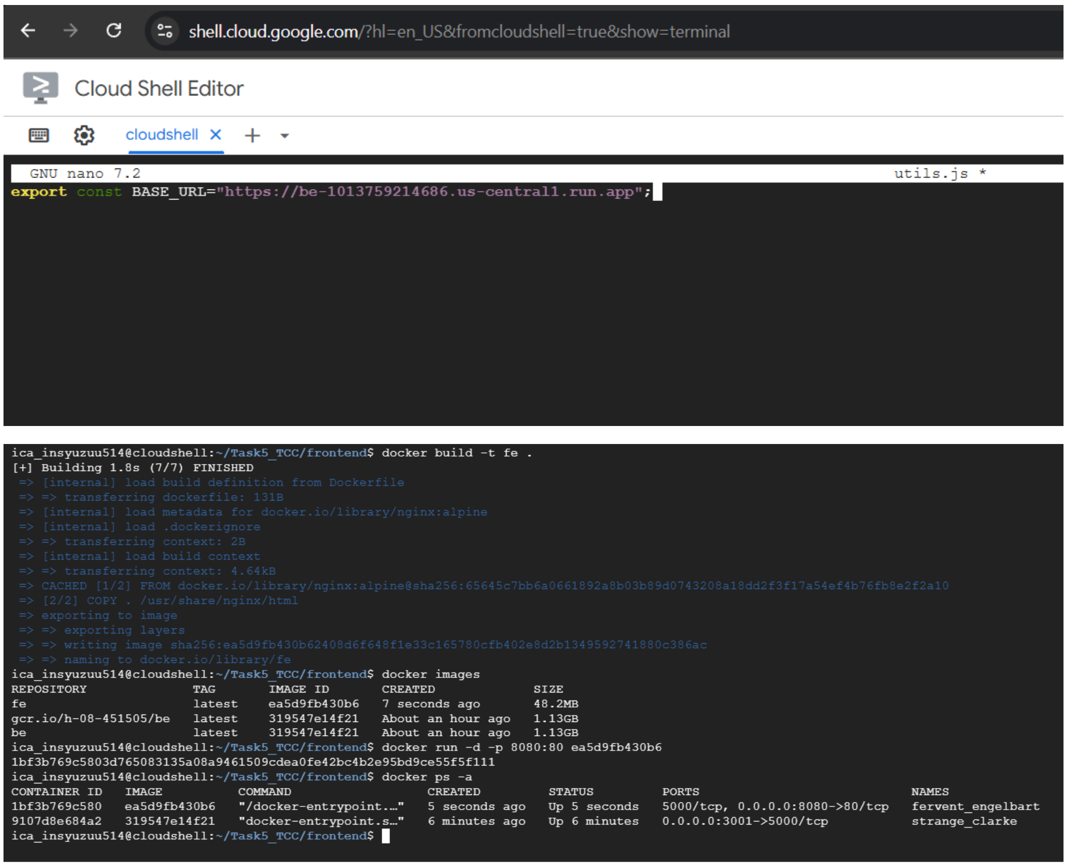
| | |
|---|---|
| Billing | Request-based |
| Startup CPU boost | Enabled |
| Concurrency | 80 |
| Request timeout | 3600 seconds |
| Execution environment | Default |

Autoscaling

| | |
|---|---|
| Revision max instances | 10 |

| | |
|---|---|
| Image | gcr.io/h-08-451505/be@sha256:6e4504070370ca... |

*Figure 4.1.5 Success Back-end*



shell.cloud.google.com/?hl=en_US&fromcloudshell=true&show=terminal

```
  GNU nano 7.2                                                        utils.js *
export const BASE_URL="https://be-1013759214686.us-central1.run.app";
```

```
ica_insyuzuu514@cloudshell:~/Task5_TCC/frontend$ docker build -t fe .
[+] Building 1.8s (7/7) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 131B
 => [internal] load metadata for docker.io/library/nginx:alpine
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load build context
 => => transferring context: 4.64kB
 => CACHED [1/2] FROM docker.io/library/nginx:alpine@sha256:65645c7bb6a0661892a8b03b89d0743208a18dd2f3f17a54ef4b76fb8e2f2a10
 => [2/2] COPY . /usr/share/nginx/html
 => exporting to image
 => => exporting layers
 => => writing image sha256:ea5d9fb430b62408d6f648f1e33c165780cfb402e8d2b1349592741880c386ac
 => => naming to docker.io/library/fe
ica_insyuzuu514@cloudshell:~/Task5_TCC/frontend$ docker images
REPOSITORY             TAG       IMAGE ID       CREATED             SIZE
fe                     latest    ea5d9fb430b6   7 seconds ago       48.2MB
gcr.io/h-08-451505/be  latest    319547e14f21   About an hour ago   1.13GB
be                     latest    319547e14f21   About an hour ago   1.13GB
ica_insyuzuu514@cloudshell:~/Task5_TCC/frontend$ docker run -d -p 8080:80 ea5d9fb430b6
1bf3b769c5803d765083135a08a9461509cdea0fe42bc4b2e95bd9ce55f5f111
ica_insyuzuu514@cloudshell:~/Task5_TCC/frontend$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS                               NAMES
1bf3b769c580   ea5d9fb430b6   "/docker-entrypoint.…"   5 seconds ago   Up 5 seconds   5000/tcp, 0.0.0.0:8080->80/tcp      fervent_engelbart
9107d8e684a2   319547e14f21   "docker-entrypoint.s…"   6 minutes ago   Up 6 minutes   0.0.0.0:3001->5000/tcp              strange_clarke
ica_insyuzuu514@cloudshell:~/Task5_TCC/frontend$
```

*Figure 4.1.6 Build docker for front end*

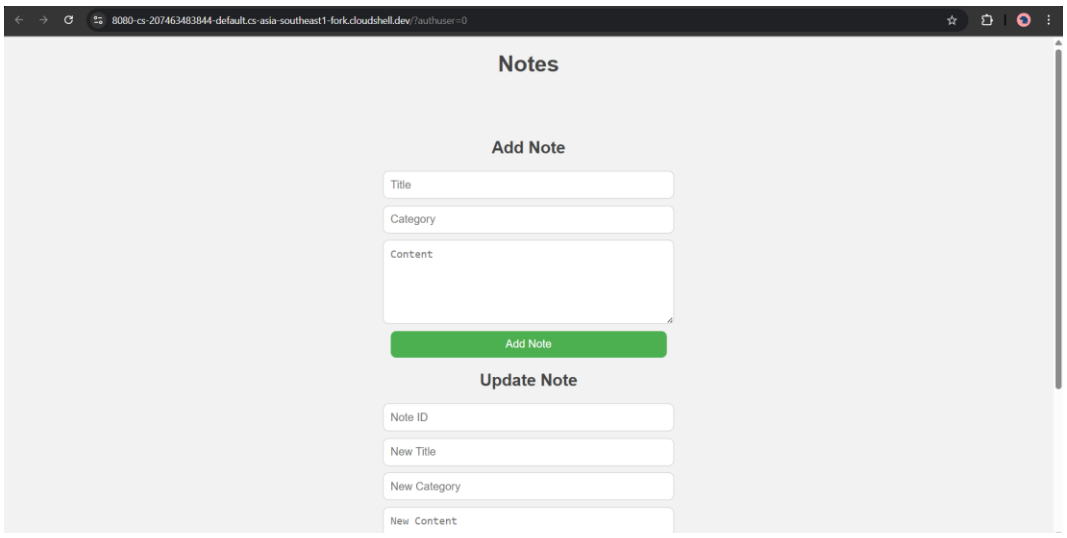


*Figure 4.1.7 Proof note app successs*

## 4.2 Discussion

The client-server architecture effectively separated the concerns of front-end and back-end, facilitating easier maintenance and updates. Using Docker for both the front-end and back-end ensured that the application could run consistently in any environment. Cloud Run provided a cost-effective and scalable solution for the back-end, while deploying the front-end via Google Cloud allowed for flexibility and ease of testing without the need for additional cloud services. However, some challenges arose in ensuring proper configuration and environment variables to allow seamless communication between the front-end and back-end services.

# CHAPTER V
# CLOSING

## 5.1 Conclusion

In conclusion, the project demonstrated the effective use of containerization, cloud deployment, and client-server architecture to build a scalable and maintainable notes application. By deploying the back-end on Cloud Run and running the front-end on Google Cloud, the system was able to handle fluctuating traffic and provide a reliable service. The project showcased the importance of modern web application practices, such as using Docker and serverless architectures.

## 5.2 Recommendation

To further improve the performance of the system, optimizing API calls between the front-end and back-end is recommended to reduce latency and enhance the overall user experience. Implementing Continuous Deployment (CI/CD) would also streamline the development process by automating deployments, ensuring quicker updates without the need for manual intervention. Additionally, future work could explore the use of more Google Cloud services, such as Firebase or App Engine, to add more comprehensive functionalities and expand the capabilities of the application. Lastly, security enhancements should be prioritized, including the implementation of HTTPS, proper authentication mechanisms, and encryption techniques to safeguard user data and prevent unauthorized access to the system.

## REFERENCES

Raj, P., & Chelladhurai, P. (2021). Containerization and Kubernetes: A comprehensive approach to cloud-native application deployment. *Journal of Cloud Computing*, 10(1), 1-15. https://doi.org/10.1186/s13677-021-00234-w