Jaypee Institute of Information Technology, Noida

Project Title: Genetic Algorithm for Optimizing Job Scheduling in Manufacturing Systems

Instructor's Name:	**Enrollment No.:**	**Name of Student:**
Dr. Goldie Gabrani	22203008	Kirti Sharma
	22203009	Shreyansh Chaudhary

Course Name: **Cloud Computing Lab**
Course Code: **25B55CS362**
Program: BSc(H)
3rd Year (6th Sem)
2025-2026

TABLE OF CONTENTS

Abstract

Task management systems are essential tools in personal productivity and project tracking. This project presents a basic web-based To-Do List application using the Flask micro web framework and containerized with Docker. The goal is to demonstrate how modern web development practices, when combined with DevOps tools, can help in creating scalable, maintainable, and portable applications. The application features a user interface for adding and deleting tasks and showcases the rapid prototyping abilities of Flask. Docker allows the application to be bundled with all its dependencies, ensuring consistent behavior across different environments. This report discusses the architecture, implementation, benefits of containerization, and potential future improvements.

Keywords

Flask, Docker, To-Do List, Web Development, Python, Containerization, HTML, CSS, CRUD Application

---

I. Introduction

In the digital era, efficient task management plays a vital role in boosting productivity, reducing procrastination, and ensuring timely execution of goals. From students managing homework to professionals handling project milestones, task management systems help keep track of responsibilities. Traditional to-do lists were often pen-and-paper-based, but today's software-driven approaches provide more interactive and efficient tools.

The rise of minimalistic yet functional web applications has encouraged the use of micro frameworks such as Flask. Flask, being lightweight and modular, is particularly suited for smaller projects or prototypes, enabling developers to quickly iterate and deploy web services. Simultaneously, Docker has revolutionized the deployment process by allowing developers to package an application with all its dependencies into a single container, ensuring consistency across different environments.

In this project, we create a simple web-based To-Do List that allows users to add and delete tasks. Though the application doesn't use a database, it lays the foundation for understanding CRUD operations and introduces containerization. It also reflects current industry practices where modular applications are containerized and deployed efficiently using Docker. This project is both educational and practical, giving insights into modern web development, containerization, and deployment workflows.

II. Literature Review

The development of web applications has seen a major shift from monolithic structures to lightweight microservices and serverless architectures. Flask, a Python micro web framework developed by Armin Ronacher, is widely appreciated for its simplicity and flexibility, especially when building small-to-medium applications [1]. Flask operates on the principles of minimalism and extensibility, making it ideal for educational projects and rapid prototyping. According to Grinberg [1], Flask allows developers to get started quickly without the burden of dependencies and boilerplate code typical in full-stack frameworks.

Simultaneously, Docker, an open-source platform introduced in 2013, has emerged as a game-changer in software deployment. Docker containers encapsulate applications along with their environments, making them platform-independent and highly portable. Merkel [2] emphasized Docker's ability to reduce environmental inconsistencies, which was a major challenge in software deployment cycles. Docker not only simplifies deployment but also plays a key role in Continuous Integration and Continuous Deployment (CI/CD) workflows.

Containerization, as discussed by Pahl [3], has enhanced the adoption of cloud-native development. It promotes modularity, scalability, and isolation between services. When paired with lightweight frameworks like Flask, Docker enables rapid and stable deployment of microservices. Further, CRUD (Create, Read, Update, Delete) operations form the backbone of most web applications, and understanding these operations is essential for any developer. Segal [4] explains that CRUD applications help learners grasp the interaction between front-end and back-end systems.

This literature highlights that combining Flask and Docker is an excellent approach for both learning and building practical applications. It also illustrates that even basic tools like a To-Do List can teach vital software engineering principles, including web development, RESTful services, containerization, and modern deployment strategies.

References:

[1] M. Grinberg, *Flask Web Development*, O'Reilly Media, 2018.
[2] D. Merkel, "Docker: Lightweight Linux Containers for Consistent Development and Deployment," *Linux Journal*, 2014(239), 2.
[3] Pahl, C., "Containerization and the PaaS Cloud," *IEEE Cloud Computing*, vol. 2, no. 3, pp. 24-31, May 2015.
[4] B. Segal, "A Short Introduction to CRUD," *ACM Ubiquity*, 2009.

III. System Architecture

A. Front-End Design

The front-end of the application is implemented using HTML and CSS, providing a visually appealing and responsive user interface. Key features include:

- Input field for task entry.
- Gradient background and modern UI design.
- Dynamic rendering of tasks using Flask's Jinja2 templating engine.

B. Back-End Logic

The back-end is written in Python using the Flask framework. Core features:

- index() function displays tasks.
- add_todo() appends a task to a list.
- delete_todo(index) removes a task by its index.
- Data is stored in a global Python list (todos[]), which is volatile and resets upon server restart.

C. Docker Integration

The application is containerized using Docker, which includes:

- A Dockerfile that specifies the Python base image and dependencies.
- Port mapping to allow external access (0.0.0.0).
- Simplified deployment using Docker commands:

```
docker build -t flask-todo .
docker run -p 5000:5000 flask-todo
```

D. File Structure

```
project/
| │-- app.py
| │-- templates/
| │    └-- index.html
| │-- Dockerfile
```

IV. Code Explanation

app.py

This script initializes the Flask application and manages HTTP routes:

```
from flask import Flask, render_template, request, redirect

app = Flask(__name__)
todos = []

@app.route('/')
def index():
    return render_template('index.html', todos=todos)

@app.route('/add', methods=['POST'])
def add_todo():
    todo = request.form.get('todo')
    if todo:
        todos.append(todo)
    return redirect('/')

@app.route('/delete/<int:index>')
def delete_todo(index):
    if 0 <= index < len(todos):
        todos.pop(index)
    return redirect('/')

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0')
```

index.html

The HTML template displays the UI, using Jinja2 syntax to loop through tasks:

```
<form action="/add" method="POST">
   <input type="text" name="todo" placeholder="What needs to be done?">
   <button type="submit">Add Task</button>
</form>

<ul>
   {% for todo in todos %}
   <li>{{ todo }} <a href="/delete/{{ loop.index0 }}">Delete</a></li>
   {% endfor %}
</ul>
```

V. Results and Testing

The application was tested on local environments using the following steps:

1. Build Docker image: docker build -t flask-todo .
2. Run container: docker run -p 5000:5000 flask-todo
3. Access the app at http://localhost:5000

Observations:

- Tasks are added and removed dynamically.
- The UI is responsive and functional across devices.
- Restarting the server resets the list due to in-memory storage.

VI. Limitations

- No persistent storage: data is lost on refresh or restart.
- Single-user environment.
- Lack of authentication.

VII. Future Scope

- Integrate database (SQLite/PostgreSQL) for persistence.
- Add user authentication.
- Improve UI using JavaScript or Bootstrap.
- Deploy to cloud using CI/CD pipelines (e.g., GitHub Actions + Heroku).

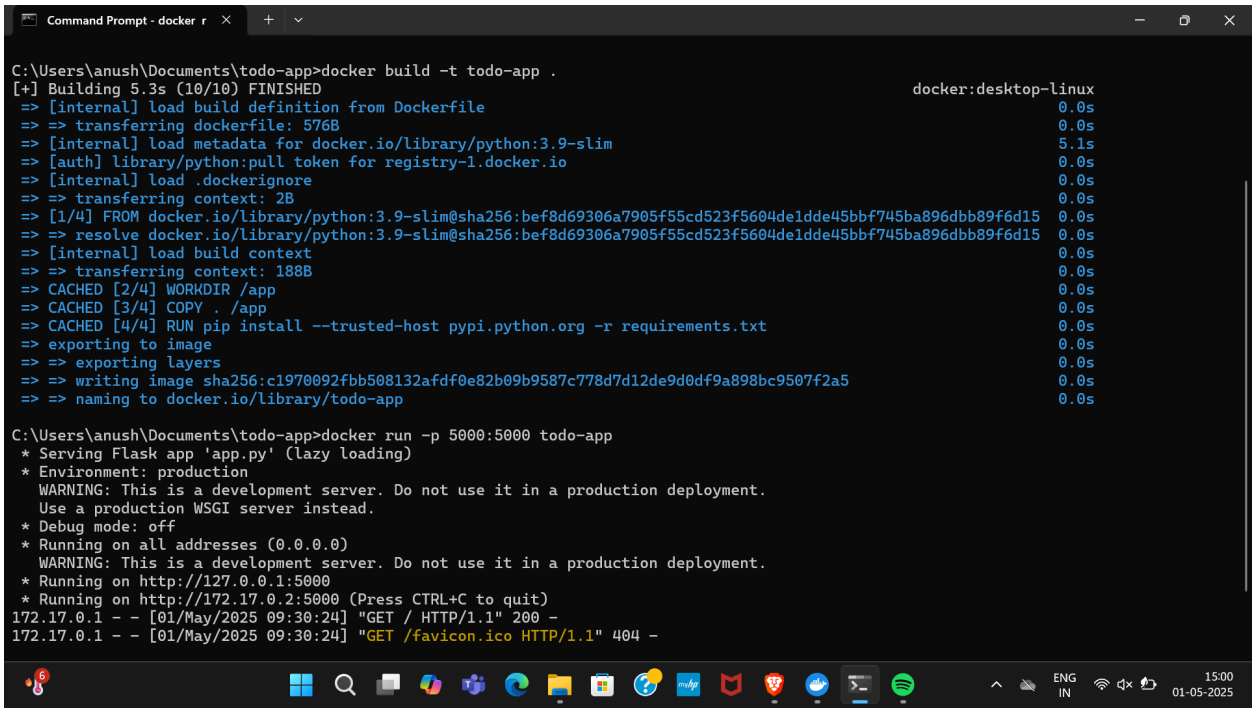
VIII. Conclusion

This project successfully demonstrates how a simple Flask application can be built, styled, and containerized using Docker. While minimal in scope, it encapsulates essential concepts in full-stack development and DevOps practices. It serves as a stepping stone toward developing scalable and maintainable web applications.

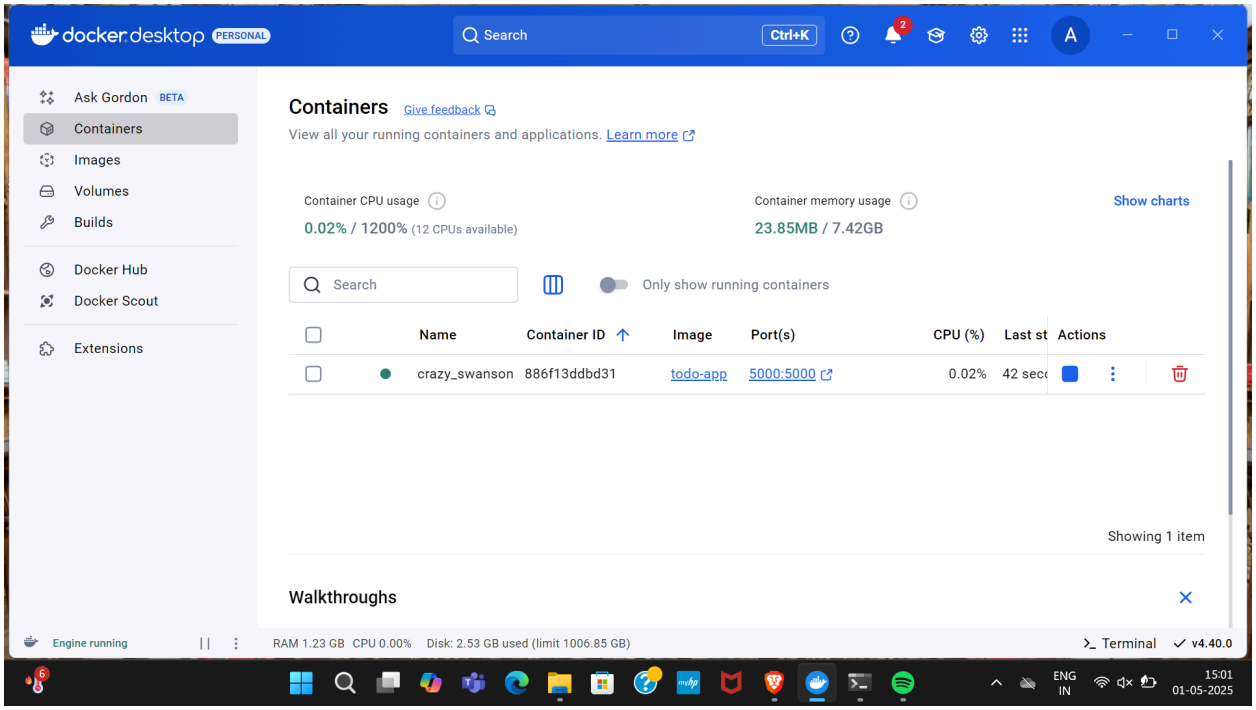
References

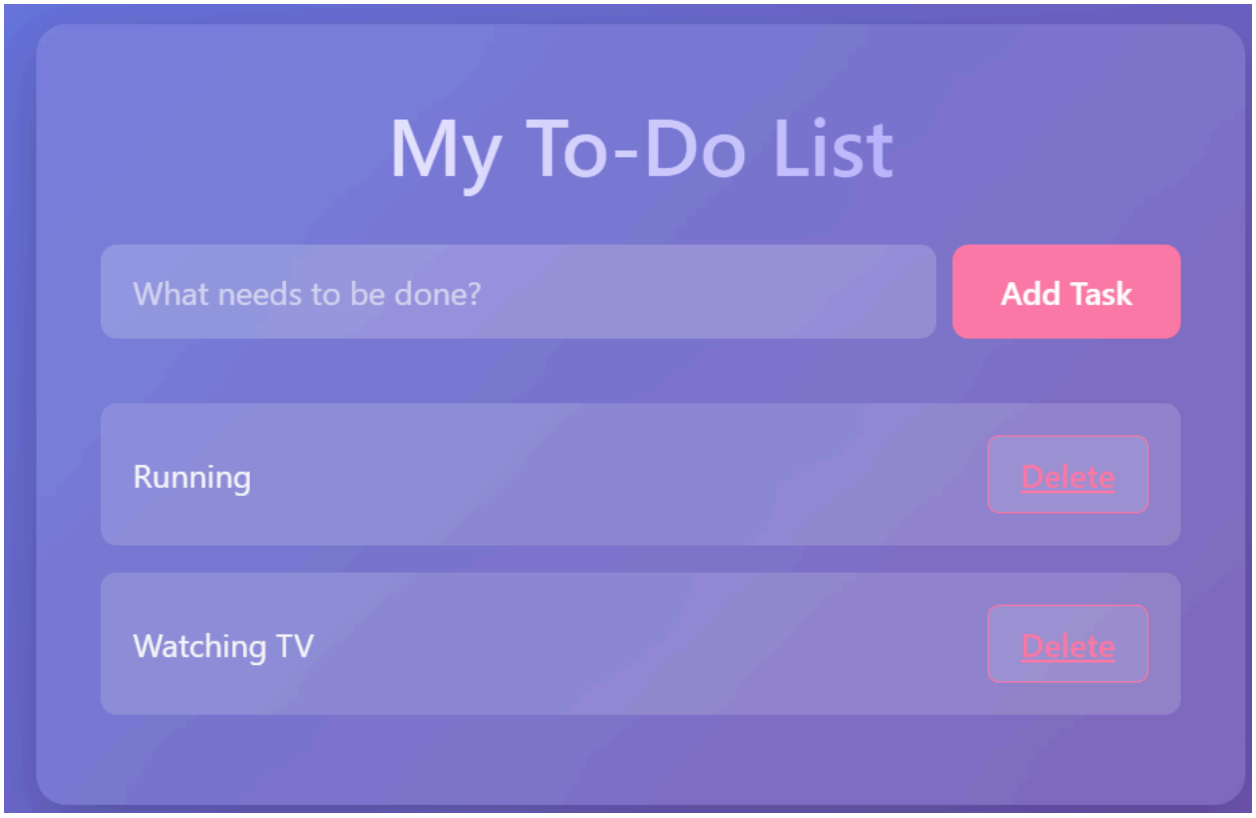
[1] M. Grinberg, *Flask Web Development*, 2nd ed., O'Reilly Media, 2018.
[2] D. Merkel, "Docker: Lightweight Linux Containers for Consistent Development and Deployment," *Linux Journal*, vol. 2014, no. 239, pp. 2, Mar. 2014.
[3] C. Pahl, "Containerization and the PaaS Cloud," *IEEE Cloud Computing*, vol. 2, no. 3, pp. 24-31, 2015.
[4] B. Segal, "A Short Introduction to CRUD," *ACM Ubiquity*, 2009.
[5] Flask Documentation. [Online]. Available: https://flask.palletsprojects.com
[6] Docker Docs. [Online]. Available: https://docs.docker.com

Appendices


Command Prompt - docker r
C:\Users\anush\Documents\todo-app>docker build -t todo-app .
[+] Building 5.3s (10/10) FINISHED docker:desktop-linux
=> [internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 576B 0.0s
=> [internal] load metadata for docker.io/library/python:3.9-slim 5.1s
=> [auth] library/python:pull token for registry-1.docker.io 0.0s
=> [internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [1/4] FROM docker.io/library/python:3.9-slim@sha256:bef8d69306a7905f55cd523f5604de1dde45bbf745ba896dbb89f6d15 0.0s
=> => resolve docker.io/library/python:3.9-slim@sha256:bef8d69306a7905f55cd523f5604de1dde45bbf745ba896dbb89f6d15 0.0s
=> [internal] load build context 0.0s
=> => transferring context: 188B 0.0s
=> CACHED [2/4] WORKDIR /app 0.0s
=> CACHED [3/4] COPY . /app 0.0s
=> CACHED [4/4] RUN pip install --trusted-host pypi.python.org -r requirements.txt 0.0s
=> exporting to image 0.0s
=> => exporting layers 0.0s
=> => writing image sha256:c1970092fbb508132afdf0e82b09b9587c778d7d12de9d0df9a898bc9507f2a5 0.0s
=> => naming to docker.io/library/todo-app 0.0s
C:\Users\anush\Documents\todo-app>docker run -p 5000:5000 todo-app
* Serving Flask app 'app.py' (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: off
* Running on all addresses (0.0.0.0)
WARNING: This is a development server. Do not use it in a production deployment.
* Running on http://127.0.0.1:5000
* Running on http://172.17.0.2:5000 (Press CTRL+C to quit)
172.17.0.1 - - [01/May/2025 09:30:24] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [01/May/2025 09:30:24] "GET /favicon.ico HTTP/1.1" 404 -
ENG
IN
15:00
01-05-2025


Microsoft Windows [Version 10.0.26100.3915]
(c) Microsoft Corporation. All rights reserved.
C:\Users\anush>cd "C:\Users\anush\Documents\todo-app"
C:\Users\anush\Documents\todo-app>for /f "tokens=*" %i in ('docker ps -aq') do docker rm %i
C:\Users\anush\Documents\todo-app>docker rmi todo-app
Untagged: todo-app:latest
Deleted: sha256:c1970092fbb508132afdf0e82b09b9587c778d7d12de9d0df9a898bc9507f2a5


docker desktop
PERSONAL
Search
Ctrl+K
Ask Gordon
BETA
Containers
Images
Volumes
Builds
Docker Hub
Docker Scout
Extensions
Containers
Give feedback
View all your running containers and applications. Learn more
Container CPU usage
0.02% / 1200% (12 CPUs available)
Container memory usage
23.85MB / 7.42GB
Show charts
Search
Only show running containers
Name
Container ID
Image
Port(s)
CPU (%)
Last st
Actions
crazy_swanson
886f13ddbd31
todo-app
5000:5000
0.02%
42 seco
Showing 1 item
Walkthroughs
Engine running
RAM 1.23 GB CPU 0.00%
Disk: 2.53 GB used (limit 1006.85 GB)
Terminal
v4.40.0
ENG
IN
15:01
01-05-2025


My To-Do List
What needs to be done?
Add Task
Running
Delete
Watching TV
Delete