Docker Flask Application Report

This report documents the process of building and running a Flask web application inside a Docker container. The project involves containerizing a simple Flask app that displays the message "Hello from Docker!" using Python 3.9 on an Alpine base image.

Project Structure:
- **app.py**: Contains the Flask web application code.
- **Dockerfile**: Defines the steps to build the Docker image.
- **requirements.txt**: Lists the dependencies (Flask).

Dockerfile Explanation:
1. Uses a lightweight base image: **python:3.9-alpine**
2. Sets the working directory to **/app** inside the container.
3. Copies **requirements.txt** and installs dependencies using pip.
4. Copies the application code to the container.
5. Exposes port **5000** (Flask default).
6. Defines the start command: **CMD ["python", "app.py"]**.

Flask App (app.py) Overview:
The app creates a simple web route at “/” that returns the message **"Hello from Docker!"**.
The application runs on host 0.0.0.0 and port 5001 to allow external access from the container.

Docker Commands Used:
1. **docker build -t test.task1 .** — Builds the Docker image.
2. **docker run -d -p 5001:5001 test.task1** — Runs the container in detached mode, mapping port 5001.
3. **docker ps** — Lists running containers.
4. **docker stop [CONTAINER_ID]** — Stops a running container.

Common Issues Encountered:
- “**OCI runtime create failed**” errors occurred due to improper image references.
- “**Port already allocated**” errors were fixed by freeing port 5000 using **sudo lsof -i :5000** and **kill** commands.
- After rebuilding, the container successfully ran and exposed the Flask app on port 5001.

Conclusion:
The Flask application was successfully containerized and deployed using Docker. The final container runs correctly on port 5001 and displays the message "Hello from Docker!" when accessed via a web browser.