**SWE40006 – Deployment Activity 4**

**Student Details**

* **Full name:** Dang Khoa Le
* **Student ID:** 103844421
* **Due date:** 28/09/2025
* **Submission date:** 24/09/2025

**Unit Details**

* **Unit code & name:** SWE40006 – Software Deployment and Evolution
* **Semester:** Semester 2, Year 2025

**Declaration of Task Levels Attempted**

* **Task 4.1 (Pass)** – Create Docker account and Docker setups.
* **Task 4.2 (Credit)** – Deploy a Hello World Python app with runtime. Pull app to another Docker device.
* **Task 4.3 (Distinction)** – Develop a web-app and deploy to Docker installation.
* **Task 4.4 (High Distinction)** – Develop a non-web-app and deploy to Docker installation.

**Prerequisites**

* **VS Code** (preferred IDE for MacOS)
* **Docker app installation and CLI**
* **Python 3.11 Runtime** (Compatible with local runtime)

Table of Contents

[4.1 Pass: Docker Setup & Hello-world 3](#_Toc209614145)

[4.1.1 Create Docker Account 3](#_Toc209614146)

[4.1.2 Start Docker Client 4](#_Toc209614153)

[4.1.3 Setups 5](#_Toc209614166)

[4.2 Credit: Python Web “Hello” + Pull to Another Device 7](#_Toc209614183)

[4.2.1 Program Topic 7](#_Toc209614184)

[4.2.2 Project Structure 7](#_Toc209614186)

[4.2.3 Build, Run, Test Locally 8](#_Toc209614216)

[4.2.4 Push to Docker Hub & Pull on Another Device 9](#_Toc209614226)

[4.3 Distinction: Web App - “SysMini” Metrics API + HTML Panel 11](#_Toc209614243)

[4.3.1 Program Topic 11](#_Toc209614244)

[4.3.2 Project Structure 11](#_Toc209614250)

[4.3.3 Build, Run, Validate 13](#_Toc209614304)

[4.4 High Distinction: Non-Web App - “csvstats” CLI 14](#_Toc209614316)

[4.4.1 Program Topic 14](#_Toc209614317)

[4.4.2 Project Structure 14](#_Toc209614319)

[4.4.3 Build & Use 16](#_Toc209614356)

[Source Coding & Brief Explanations 18](#_Toc209614384)

[Appendix 18](#_Toc209614415)

**4.1 Pass: Docker Setup & Hello-world**

**4.1.1 Create Account**

**4.1.1.1 Credentials**

1. Go to Docker Hub and sign up with a student email (optinal).
2. Verify email and sign in.

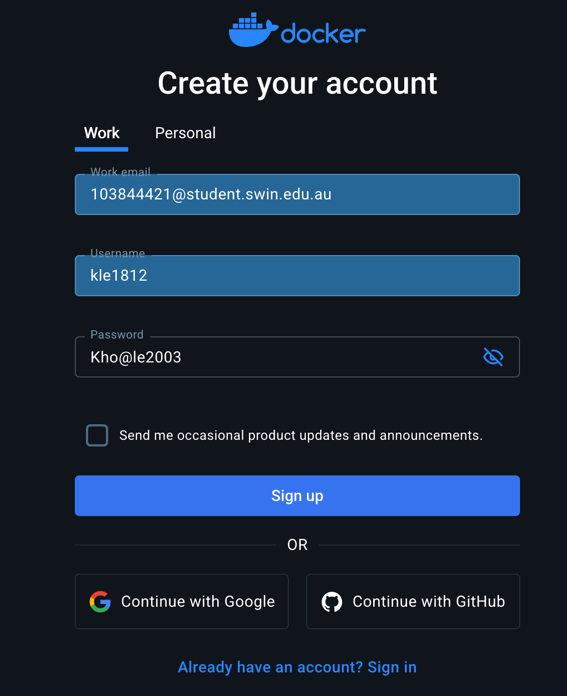


Figure 1. Docker account registration with student email

A screenshot of a login page

Description automatically generated

Figure 2. Docker account signin with student email

1. There will be a mail send to the (student) email registered with this Docker Hub account, where we need to click to “Verify Email Address” to verify account.

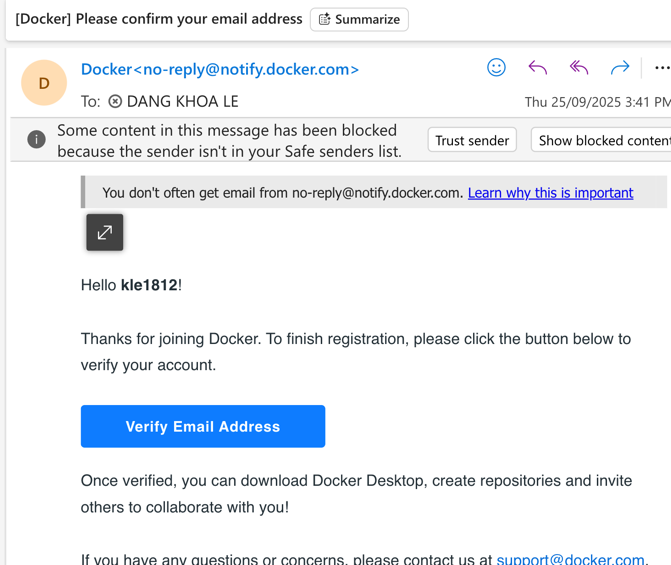


Figure 3. Account email verification

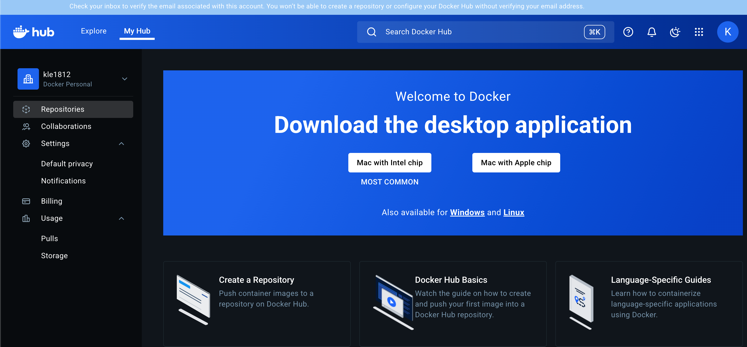


Figure 4. Docker Hub Dashboard

**4.1.1.2 Installation**

* I already downloaded Docker application previously and I am using ‘Mac with Intel chip’ version package.

**4.1.2 Start Docker Client**

1. Pre-requisites: Install Docker Desktop. Follow any installation walkthrough.
2. Open Docker application, sign in with pre-created account.
3. Start Docker from the application UI or follow CLI command:

sudo systemctl start docker

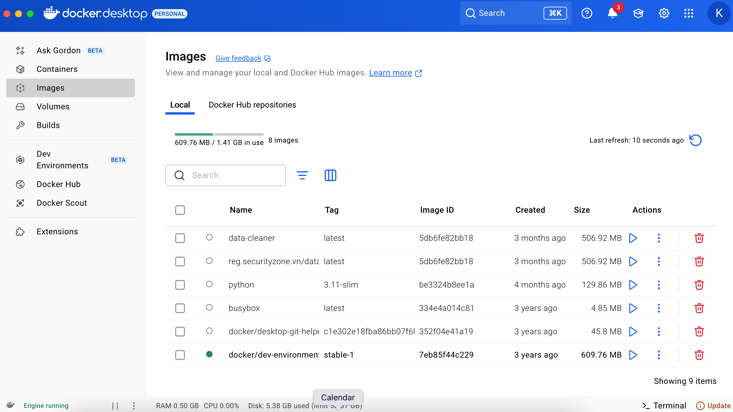


Figure 5 Docker desktop app with my existing images.

1. My application already enabled option “Start Docker Desktop when you sign in to your computer”, so Docker automatically started once we sign in or open the desktop app. This could be configured from the Setting page (gear icon) on the navigation bar.

A screenshot of a computer

Description automatically generated

Figure 6. Docker desktop app with “Start Docker Desktop when you sign in to your computer” option.

If we want Docker to start automatically at boot, we can enable the service by CLI command:

sudo systemctl enable docker

**4.1.3 Setups**

Run CLI to verify installation and downloaded version:

docker --version

docker info

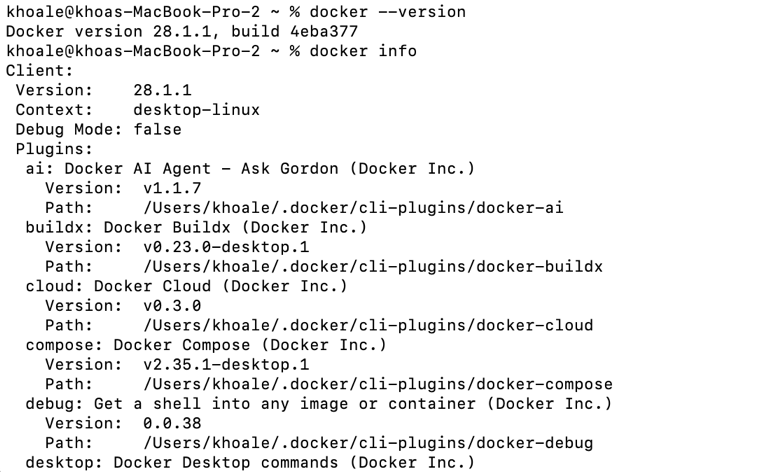
****

Figure 7. Docker package verification

This shown that we have docker version *28.1.1*.

We can login to Docker Hub via CLI commands as well, this will allow us to push any app’s image to Docker Hub later on:

docker login

**A screenshot of a computer error

Description automatically generated**

Figure 8. Docker login CLI

Which will require us to activate our device via a browser URL. Enter the device confirmation code there.

A screenshot of a device registration

Description automatically generated

Figure 9. Docker device activation

We can also create a *Personal Access Token (PAT)* from Docker Hub, then next time login, we can prompt that PAT token to password form. The access token will be in form of: dckr\_pat\_. The access token can have customised permission that different action can be granted according to the task.

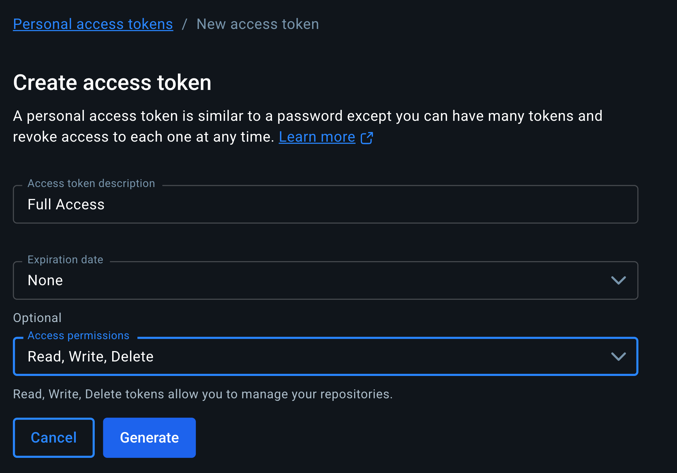


Figure 10. PAT generation

**4.2 Credit: Python Web “Hello” + Pull to Another Device**

**4.2.1 Program Topic**

We’ll deploy a minimal **Flask** web app (slightly above trivial “hello-world”): it exposes / with a JSON greeting and /health for health checks. This establishes a correct Python runtime and a working containerized web service.

**4.2.2 Project Structure**

web-hello/

├─ app.py

├─ requirements.txt

└─ Dockerfile

**app.py:**

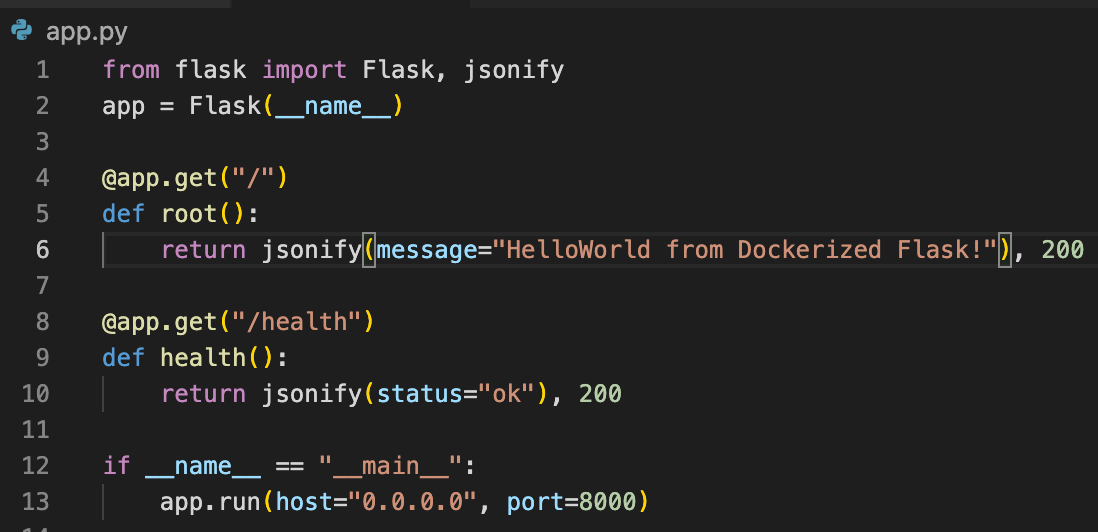
****

Figure 11. Hello webapp’s app.py – built with Flask

The application is a lightweight Flask-based web server that exposes two endpoints:

**/ (root)** → Returns a JSON message:

{"message": "HelloWorld from Dockerized Flask!"}

**/health** → Returns a health-check JSON response:

"status": "ok"}

The app will run on port 8000 (default Python app’s port), serving on 0.0.0.0 host (IP block containing all possible IP addresses or everywhere access)

**requirements.txt:**

Flask is specified as a runtime dependency in requirements.txt with a specific version matching our local device:

flask==3.0.3

**Dockerfile:**

Beforehand, I will check my current Python version from local device, from CLI command:

python3 –version

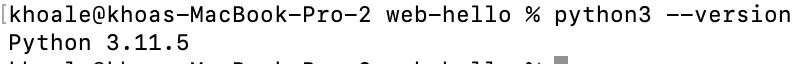


Figure 12. Local Python runtime version

Since our device is using Python3.11 runtime, I will proceed with all of other tasks with Python3.11 image configuration and build:

A screen shot of a computer program

Description automatically generated

Figure 13. Hello webapp’s Dockerfile

The app is containerized with a Dockerfile that:

1. Uses the official *python:3.11 -slim* image (small, secured).
2. Sets the working directory (/app).
3. Installs Flask from requirements.txt.
4. Copies in the source code and runs the app on port **8000**.

This design makes the app portable and easy to run consistently across environments. The hello endpoint demonstrates basic JSON serving, while /health can be used by orchestration systems (e.g., Docker Compose, Kubernetes) to check if the container is running correctly.

**4.2.3 Build, Run, Test Locally**

We build our Dockerized app with the Docker Hub username (kle1812) and specified the first version as 1.0. Then we run following command to run the web-hello app. Actions can be executed by these CLI commands:

docker build -t kle1812/web-hello:1.0 .

docker run --rm -p 8000:8000 kle1812/web-hello:1.0

**A screenshot of a computer

Description automatically generated**

Figure 14. Hello webapp’s Docker build completion

A screenshot of a computer code

Description automatically generated

Figure 15. Hello webapp’s Docker image run

We will access our localhost on port 8000 to inspect the Hello web-app and use CURL request to its health endpoint to check the system health status:

curl <http://localhost:8000/health>

A screenshot of a phone

Description automatically generated

Figure 16. Hello webapp’s Docker image run on web browser

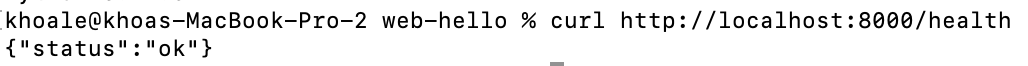


Figure 17. Hello webapp’s Docker image runtime pass health check

**4.2.4 Push to Docker Hub & Pull on Another Device**

On Docker Hub website, navigate to Repositories from sidebar navigation. Create a repository named web-hello (this app):

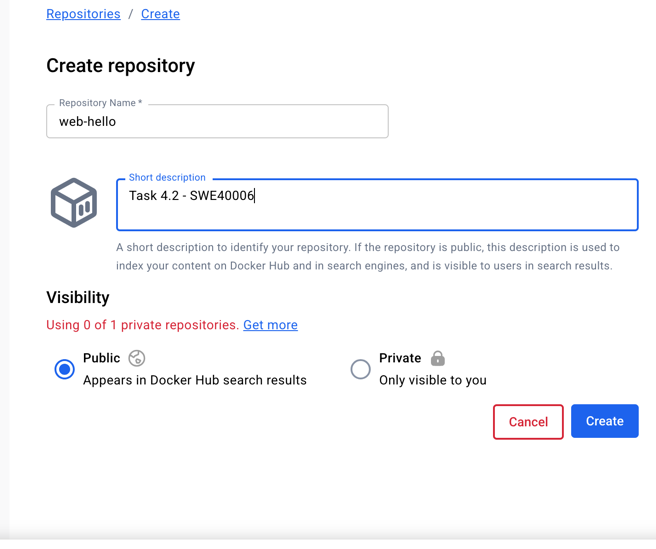


Figure 18. Docker Hub repository creation

We can run this CLI command to inspect the image and derive its id from image name:

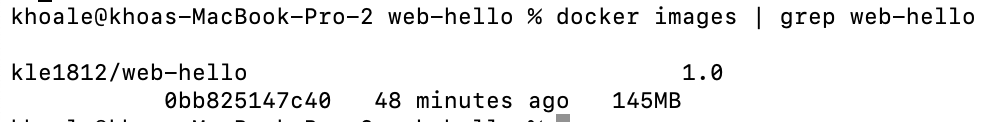


Figure 19. Docker image id identification

On this current device, we login (if haven’t done previously), tagged image version *1.0* (optional, a tag serves as an identifier for a specific image version, helps distinguish different builds or configurations of the same image), and push our app to Docker Hub, following this CLI commands:

docker push kle1812/web-hello:1.0

**A white background with black text

Description automatically generated**

Figure 20. Docker image pushed to Docker Hub

A screenshot of a computer

Description automatically generated

Figure 21. Docker image shown on Docker Hub repository

On a second machine / device, we first pull the repository to local device (we may need to attempt docker login if the repository is private), then we can run the image:

docker pull kle1812/web-hello:1.0

docker run --rm -p 8000:8000 kle1812/web-hello:1.0

However, for this task, since I don’t have another device, we can simulate this my removing the image locally with CLI command:

docker rmi kle1812/web-hello:1.0

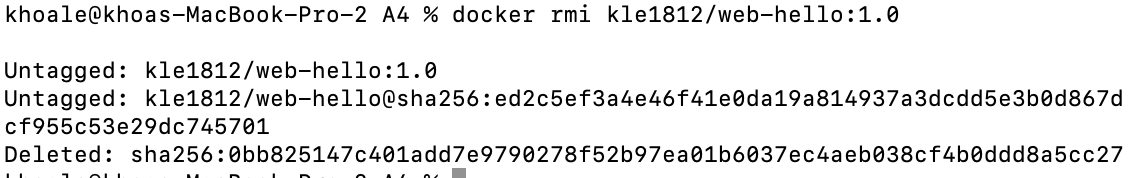


Figure 22. Docker image deletion

Then attempt to freshly pull it back from Docker Hub:

A screen shot of a computer

Description automatically generated

Figure 23. Docker image pull back

Which now I can run it again:

A screenshot of a computer code

Description automatically generated

Figure 24. Docker image run after pull back

And prove that the project still work perfectly after pulling back (tested with CURL request to localhost port 8000 where the app is running:

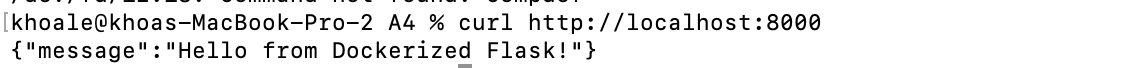


Figure 25. Docker image runtime validation after pull back

**4.3 Distinction: Web App - “SysMini” Metrics API + HTML Panel**

**4.3.1 Program Topic**

The **SysMini** application is a lightweight system monitoring service developed with **FastAPI**. Its goal is to demonstrate how a Dockerized Python web service can provide both a **dynamic web interface** and **machine-readable API endpoints**. Unlike a basic hello-world app (in 4.2), SysMini integrates **real-time system metrics** using the psutil library, which makes it more meaningful and closer to a real-world use case:

* Serves / — a tiny HTML dashboard (no heavy frontend) polling JSON, append to tabl.
* Serves /metrics — JSON for CPU%, memory%, and load averages (via psutil).
* Serves /health — returns {status:"ok"}.

**4.3.2 Project Structure**

sysmini/

├─ app.py

├─ requirements.txt

└─ Dockerfile

**app.py:**

**A screen shot of a computer screen

Description automatically generated**

Figure 26. Sysmini’s app.py core

* **Framework: FastAPI**
  + Chosen for its speed, modern async design, and easy support for both **HTML** and **JSON** responses.
  + Automatically generates OpenAPI documentation (accessible at /docs) which is useful for future extensibility.
* **Metrics Collection: psutil**
  + psutil.cpu\_percent(interval=0.2) → captures CPU usage in short intervals.
  + psutil.virtual\_memory().percent → reports memory utilization.
  + os.getloadavg() → fetches system load averages (1m, 5m, 15m). On Windows, this gracefully falls back to (0.0, 0.0, 0.0).
* **Endpoints**
  + / → Returns an HTML dashboard with a JavaScript auto-refresh script. Every 2 seconds, it polls the /metrics endpoint and displays the table.
  + /metrics → Provides live system stats in JSON format for programmatic consumption (e.g., monitoring agents, cURL, or other services).
  + /health → Lightweight health check returning {status: "ok"} for Docker/Kubernetes probes.
* **Front-end Dashboard**

A screen shot of a computer program

Description automatically generated

Figure 27. Sysmini’s app.py HTML frontend rendering

* **Minimal HTML + embedded JavaScript:** A single static page with inline <script> logic. No build tools, frameworks, or bundlers required.
* **Tabular presentation for clarity:** Metrics are displayed in a simple, responsive HTML <table>, with CPU, memory, and load averages shown in separate columns. This improves readability compared to raw JSON dumps.
* **No external libraries (keeps image small):** Styling uses plain CSS embedded in the page header, which avoids dependencies like Bootstrap or jQuery.
* **Automatic refresh every 2 seconds:** A setInterval call triggers /metrics polling and updates the <td> cells in the table dynamically, giving the user real-time feedback without page reload.

**requirements.txt:**

fastapi==0.112.2

uvicorn==0.30.5

psutil==6.0.0

* fastapi (API framework)
* uvicorn (production-grade ASGI server)
* psutil (system monitoring)

**Dockerfile:**

Merely identical to the earlier Hello web-app from section 4.2.

* Uses python:3.11-slim base (small, secure and compatible with local setup).
* Installs only necessary dependencies.
* Exposes port 8000 and launches via uvicorn.

A screen shot of a computer

Description automatically generated

Figure 28. Sysmini’s Dockerfile

**4.3.3 Build, Run, Validate**

We build our Dockerized app with the Docker Hub username (kle1812) and specified the first version as 1.0. Then we run following command to run the web-hello app. Actions can be executed by these CLI commands:

docker build -t kle1812/sysmini:1.0 .

docker run --rm -p 8000:8000 kle1812/sysmini:1.0

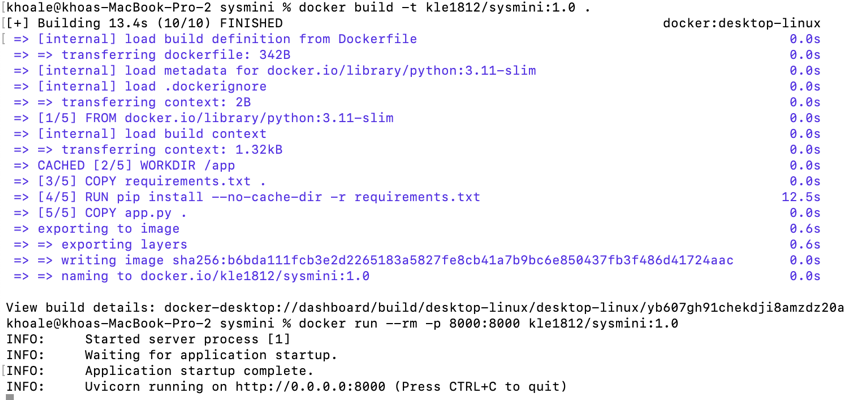


Figure 29. Sysmini’s app build and run confirmation on CLI

Open http://localhost:8000/ for the live table panel of system status.

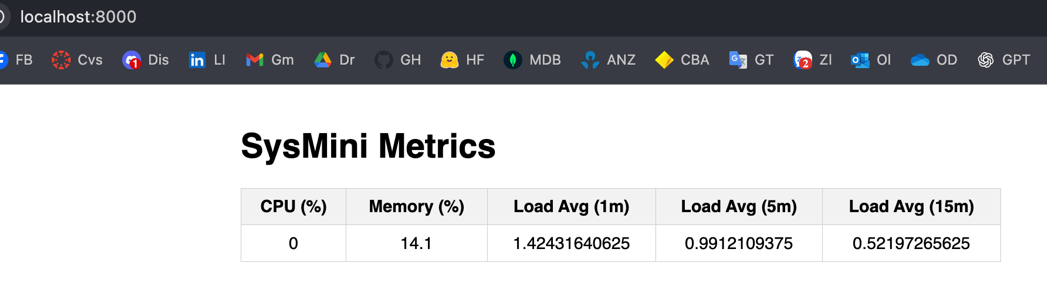


Figure 30. Sysmini’s app build and run confirmation on CLI

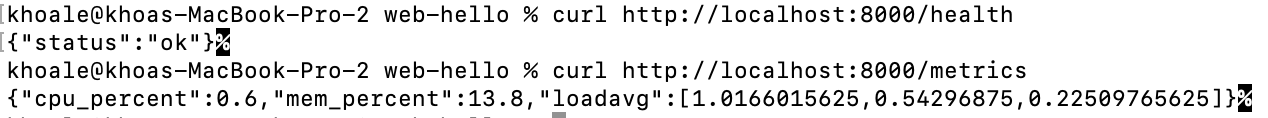
****

Figure 31. Sysmini’s app /metrics and /health endpoint tests with CURL

Optional, we can also push the app to Docker Hub repository too. Following CLI command:

docker push kle1812/sysmini:1.0

A screen shot of a computer

Description automatically generated

Figure 32. Sysmini successful push to Docker Hub repository

A screenshot of a phone

Description automatically generated

Figure 33. Inspect Sysmini from Docker Hub repository

**4.4 High Distinction: Non-Web App - “csvstats” CLI**

**4.4.1 Program Topic**

A Dockerized **command-line** Python utility that computes simple statistics (count, mean, min, max) for a numeric column in a CSV file passed via a mounted volume. This is distinctly **non-web** and demonstrates practical data handling.

The app simulating Data Analytics with proper error and exception handlings, to analyse tourism flow on selective cities using visitor data synthesis.

**4.4.2 Project Structure**

csvstats/

├─ csvstats.py

├─ requirements.txt

└─ Dockerfile

**csvstats.py**

****

Figure 34. csvstats app’s main file

The app utilise similar architecture with task 4.2 and 4.3 to containerized with a Dockerfile:

1. **Imports**

* sys handles command-line arguments and exit codes.
* pandas provides robust CSV reading and numeric operations.

1. **Usage Function**

* Displays correct usage if the user provides wrong or missing arguments.
* Exit code 1 signals improper usage.

1. **Main Execution Block**

* Ensures code only runs when executed directly, not when imported as a library.

1. **Argument Parsing**

* Expects exactly 2 arguments:
  + path → path to the CSV file.
  + col → column name to analyze.
* Calls usage() if arguments are missing.

1. **CSV Loading & Column Validation**

* Reads the CSV into a DataFrame.
* Exits with code 2 if the requested column does not exist.

1. **Data Cleaning & Conversion**

* Converts the chosen column into numeric values.
* Non-numeric entries become NaN and are dropped → ensures clean stats.

1. **Statistics Output**

* Displays:
  + Total rows in CSV.
  + Count of valid numeric values in the chosen column.
  + Minimum and maximum values.
  + Mean (average) and median (central tendency).
* Exit code 0 confirms success.

1. **Error Handling**

* Exit 3 if file is missing.
* Exit 4 for unexpected runtime errors.
* Provides clear error messages for debugging.

**Summary**

* Demonstrates a **CLI-only app**, proving Docker can package not just web servers but also **utility tools**.
* Code is simple but covers **robust input validation**, **data cleaning**, and **error handling**.
* Adds value by showing real-world relevance: analyzing data files directly inside containers, ensuring portability across environments.

**requirements.txt**

pandas==2.2.2

We install pandas (with compatible version to Python3.11 runtime), for robust CSV reading and numeric operations.

**Dockerfile**

The app utilise similar architecture with task 4.2 and 4.3 to containerized with a Dockerfile:

1. Uses the official *python:3.11 -slim* image (small, secured).
2. Sets the working directory (/app).
3. Installs Flask from requirements.txt.
4. Copies in the source code and runs the csvstats app file on port **8000**.

**4.4.3 Build & Use**

We run the following CLI command to build Docker image for this app:

docker build -t kle1812/csvstats:1.0 .

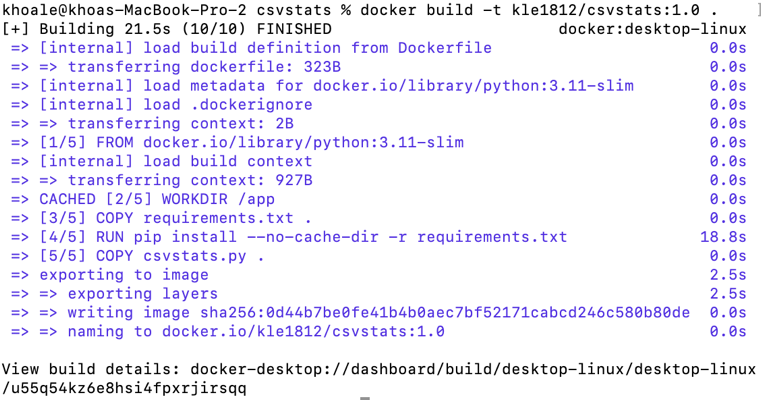


Figure 35. csvstats app’s Docker build

Create a sampled CSV data synthetically demonstrate popular city name and number of visitor, following this CLI command:

cat > /tmp/sample.csv <<'EOF'

city,visitors

Melbourne,10

Sydney,15

Hanoi,9

Hue,abc

Saigon,21

EOF

Or you can create any sample CSV that consist of 2 tables city and visitors

Run the tool (mount /tmp):

docker run --rm -v /tmp:/data kle1812/csvstats:1.0 /data/sample.csv visitors

Expected output (handles non-numeric “abc” gracefully, which will clean the data to be 0):

Rows: 5

Valid numeric in 'visitors': 4

Min: 9.0 | Max: 21.0

Mean: 13.75 | Median: 12.5

A white background with black text

Description automatically generated

Figure 36. csvstats app’s Docker run work as expected on sample.csv

Optional, we can also push the app to Docker Hub repository too. Following CLI command (tagged version 1.0):

docker push kle1812/csvstats:1.0

A screenshot of a phone

Description automatically generated

Figure 37. We are now featured 3 images on Docker Hub public repository

And pull it back with command:

docker pull kle1812/csvstats:1.0

**Source Coding & Brief Explanations**

**Why python:3.11-slim?**

* Stable wheels for Flask/FastAPI/pandas/psutil; minimal image size; faster builds.
* Clear dependency layers (requirements then code) → efficient Docker cache usage.
* Compatible with local device.

**Key Dockerfile Practices**

* WORKDIR /app ensures predictable paths.
* COPY requirements.txt → RUN pip install first to leverage caching.
* EXPOSE 8000 is documentation for orchestration; actual mapping is via -p host:container.

**Health & Observability**

* /health endpoints in both web apps for readiness checks.
* Plain JSON + curl verification for repeatable, automated tests.

**Appendix**

Docker Hub repo URL: <https://hub.docker.com/repositories/kle1812>

All code resources can be found from this Google Drive access URL: <https://drive.google.com/drive/folders/1gHFJvXgMUVQFIOXhCo6M1bOftoXQr65r?usp=sharing>