

Dockerized Python + PostgreSQL – Step-by-Step Project

Documentation

1. Introduction

This project demonstrates how to use Docker to containerize a simple Python application and connect it to a PostgreSQL database running in another container.

By the end of the project, the app:

- Connects to PostgreSQL

- Creates a table

- Inserts a row

- Reads and prints the data

Everything runs inside Docker containers using a custom Docker network.

2. Tools & Technologies Used

- Operating System: Windows 10/11 (64-bit)

- Docker Desktop (Linux containers)

- Visual Studio Code (VS Code)

- Python 3.10 (Docker image: python:3.10-slim)

- PostgreSQL (Docker image: postgres)

- Python library: psycopg2-binary (PostgreSQL driver)

3. Verifying Docker Installation

Step 3.1 – Check Docker is installed and running

Open Command Prompt or PowerShell and run:

`docker version`

This shows both Client and Server information.

- ✓ If both sections appear → Docker Desktop is installed and running correctly.

Screenshot 1:

The screenshot shows a VS Code interface with the following details:

- Explorer View:** Shows a project named "MY-APP" containing files: "app.py", "Dockerfile", and "README.md".
- Terminal View:** Displays the command `PS C:\Users\lenovo\Desktop\my-app> docker version` and its output:

```
runc:
  Version:      1.3.4
  GitCommit:    v1.3.4-0-gd6d73eb8
docker-init:
  Version:      0.19.0
  GitCommit:    de40ad0
```
- Status Bar:** Shows the path `C:\Users\lenovo\Desktop\my-app`.

4. Setting Up VS Code With Docker

Step 4.1 – Install VS Code

Download from:
<https://code.visualstudio.com/>

Step 4.2 – Install Docker Extension in VS Code

1. Open VS Code
2. Click on the **Extensions** icon on the left sidebar
3. Search for “**Docker**” (by Microsoft)
4. Click **Install**

Screenshot 2:



Step 4.3 – Verify Docker view in VS Code

1. Click the **Docker** icon on the left sidebar
2. You should see sections like:

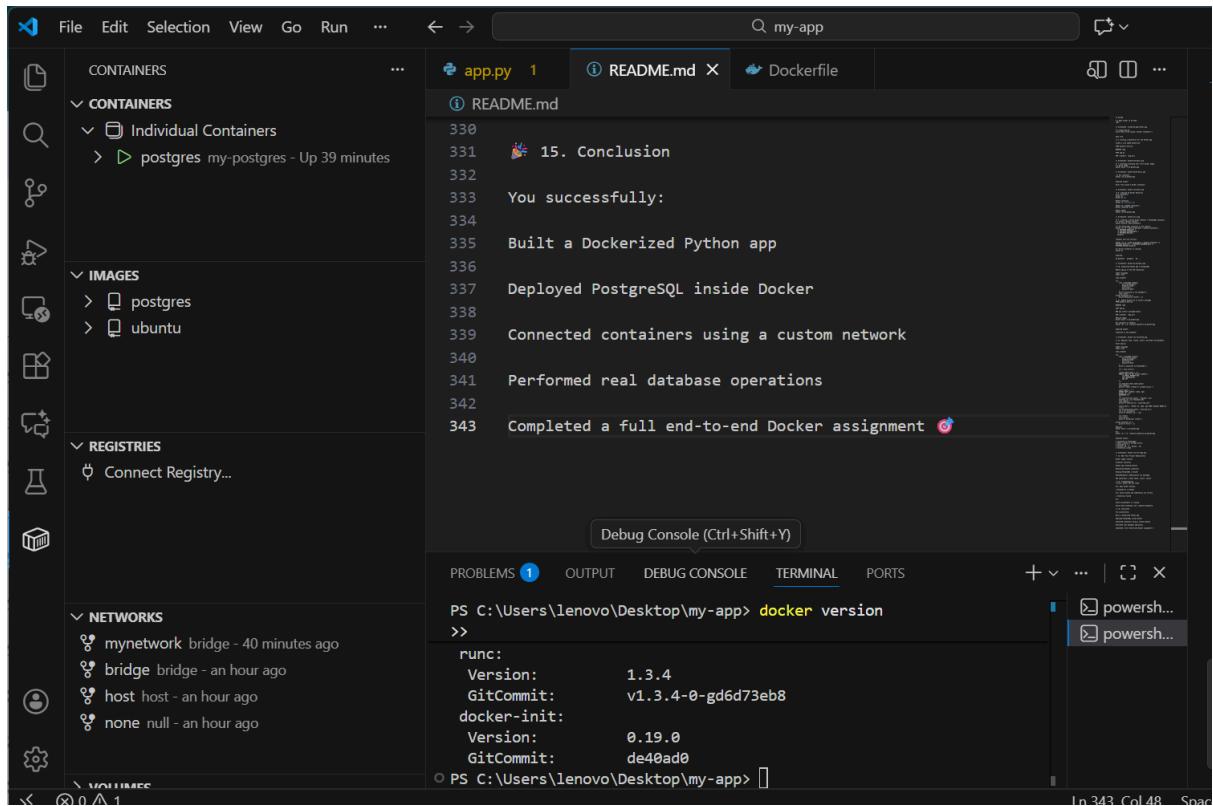
- Containers

- Images

- Volumes

- Networks

Screenshot 3:



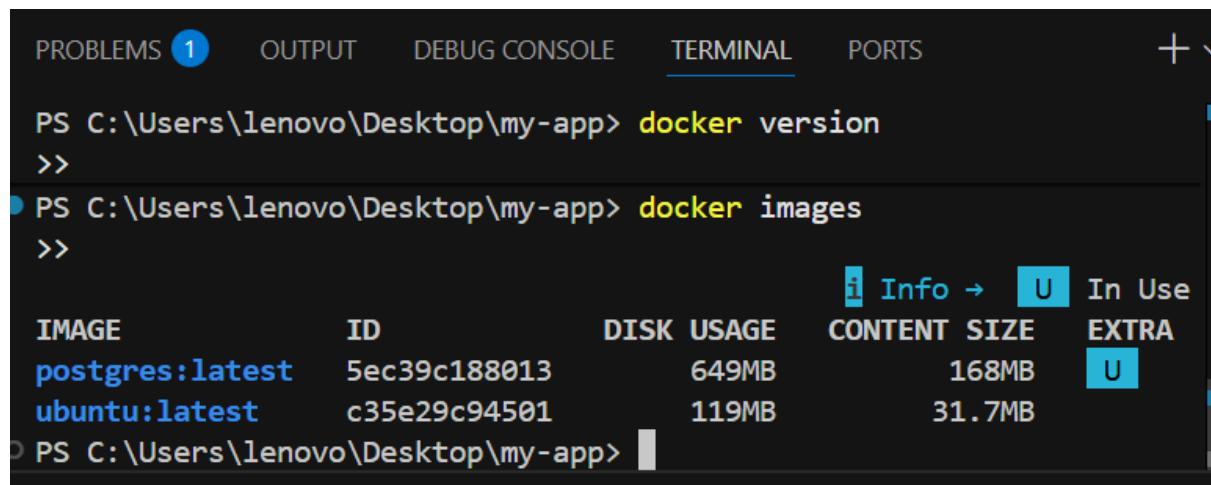
Step 5.1 – Pull a Base Image

Run the following command to pull an Ubuntu image:

```
docker pull ubuntu
```

Step 5.2 – View Downloaded Images

docker images



The screenshot shows the VS Code interface with the Terminal tab selected. The command `docker version` was run, followed by `docker images`. The output lists two images: `postgres:latest` and `ubuntu:latest`. The table includes columns for IMAGE, ID, DISK USAGE, CONTENT SIZE, and EXTRA.

IMAGE	ID	DISK USAGE	CONTENT SIZE	EXTRA
<code>postgres:latest</code>	<code>5ec39c188013</code>	<code>649MB</code>	<code>168MB</code>	<code>U</code>
<code>ubuntu:latest</code>	<code>c35e29c94501</code>	<code>119MB</code>	<code>31.7MB</code>	<code>U</code>

6. Creating a Simple Python App

Step 6.1 – Create a project folder

```
mkdir my-app  
cd my-app
```

Step 6.2 – Open the folder in VS Code

```
code .
```

Step 6.3 – Create `app.py`

Inside VS Code → New File → `app.py`

7. Creating the Dockerfile

Create a new file named **Dockerfile** (no extension):

The screenshot shows the Visual Studio Code interface with the following details:

- Explorer View:** Shows a project named "MY-APP" containing files: app.py, Dockerfile, and README.md.
- Code Editor:** The "app.py" file is open, displaying Python code for connecting to a database and executing queries. The code includes imports for sqlite3, os, and json, and defines functions for inserting and selecting data from a "students" table.
- Terminal:** The terminal tab is active, showing the command "docker images" being run in a Windows command prompt (PS C:\Users\lenovo\Desktop\my-app>). The output lists two images:

IMAGE	ID	DISK USAGE	CONTENT SIZE	EXTRA
postgres:latest	5ec39c188013	649MB	168MB	U
ubuntu:latest	c35e29c94501	119MB	31.7MB	
- Output:** The "OUTPUT" tab shows the command "code ." being run in the terminal.

8. Building the Docker Image

Run:

```
docker build -t my-python-app .
```

9. Running the Python Container

Run the container:

The screenshot shows the Docker Extension in Visual Studio Code. The left sidebar displays the Docker Explorer with sections for CONTAINERS, IMAGES, REGISTRIES, NETWORKS, and VOLUMES. The IMAGES section shows an image named 'my-python-app' selected. The right side of the interface includes a code editor with Python code for interacting with a database, a terminal window showing the output of a 'docker build' command, and a Problems panel.

```
42     # 4. Read and print the row
43     select_query = "SELECT id, name, age FROM students WHERE id ="
44     cur.execute(select_query, (inserted_id,))
45     row = cur.fetchone()
46     print("✓ Fetched row:", row)
47
48     # 5. Close everything
49     cur.close()
50     conn.close()
51     print("✓ Connection closed.")
52
53 except Exception as e:
54     print("✗ Error:", e)
```

```
PS C:\Users\lenovo\Desktop\my-app> docker build -t my-python-app .
>>
=> => exporting manifest sha256:8f17f137ba194246a361c5d1c0d 0.0s
=> => exporting config sha256:1f56a7add6bc219dd1213bf025907 0.0s
=> => exporting attestation manifest sha256:5729eeb712fc63a 0.1s
=> => exporting manifest list sha256:89884e95de35f08c7d9750 0.0s
=> => naming to docker.io/library/my-python-app:latest 0.0s
=> => unpacking to docker.io/library/my-python-app:latest 0.8s
```

Cleaning Up Docker Resources (Optional)

View containers:

Remove an image:

The screenshot shows the Docker Extension for VS Code interface. On the left, the sidebar displays:

- CONTAINERS**: Individual Containers (my-python-app, postgres), Registry (Connect Registry...).
- IMAGES**: my-python-app, postgres, ubuntu.
- REGISTRIES**: Connect Registry...
- NETWORKS**: mynetwork, bridge, host, none.
- VOLUMES**.

The main area shows the `app.py` file content:

```
42     # 4. Read and print the row
43     select_query = "SELECT id, name, age FROM students WHERE
44     cur.execute(select_query, (inserted_id,))
45     row = cur.fetchone()
46     print("✅ Fetched row:", row)
47
48     # 5. Close everything
49     cur.close()
50     conn.close()
51     print("✅ Connection closed.")
52
53 except Exception as e:
54     print("❌ Error:", e)
```

The **TERMINAL** tab at the bottom shows the command-line output:

```
PS C:\Users\lenovo\Desktop\my-app> docker ps -a
>>
ago   Up 53 minutes          5432/tcp    my-postgres
② PS C:\Users\lenovo\Desktop\my-app> docker rmi my-python-app
>>
Error response from daemon: conflict: unable to delete my-python-app:latest (must be forced) - container 556087b475be is using its referenced image 89884e95de35
△ PS C:\Users\lenovo\Desktop\my-app>
```

11. Creating a Custom Docker Network

A network allows containers to communicate with each other.

Create the network:

The screenshot shows the Docker Extension in Visual Studio Code. The left sidebar displays the Docker tree with sections for CONTAINERS, IMAGES, and REGISTRIES. The IMAGES section shows three entries: my-python-app, postgres, and ubuntu. The terminal tab at the bottom shows the following command history:

```
PS C:\Users\lenovo\Desktop\my-app> docker rmi my-python-app
>>
p:latest (must be forced) - container 556087b475be is using its referenced image 89884e95de35
④ PS C:\Users\lenovo\Desktop\my-app> docker network create mynetwork
>>
Error response from daemon: network with name mynetwork already exists
④ PS C:\Users\lenovo\Desktop\my-app>
```

12. Running PostgreSQL in a Container

Run PostgreSQL and attach it to the custom network:

The screenshot shows the Docker Extension in Visual Studio Code with the terminal tab active. The terminal output shows the following commands:

```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS +
```

```
PS C:\Users\lenovo\Desktop\my-app> docker network create mynetwork
>>
● PS C:\Users\lenovo\Desktop\my-app> docker ps
>>
CONTAINER ID IMAGE COMMAND CREATED
STATUS PORTS NAMES
2d47725c0ce5 postgres "docker-entrypoint.s..." 55 minutes ago
Up 55 minutes 5432/tcp my-postgres
④ PS C:\Users\lenovo\Desktop\my-app>
```

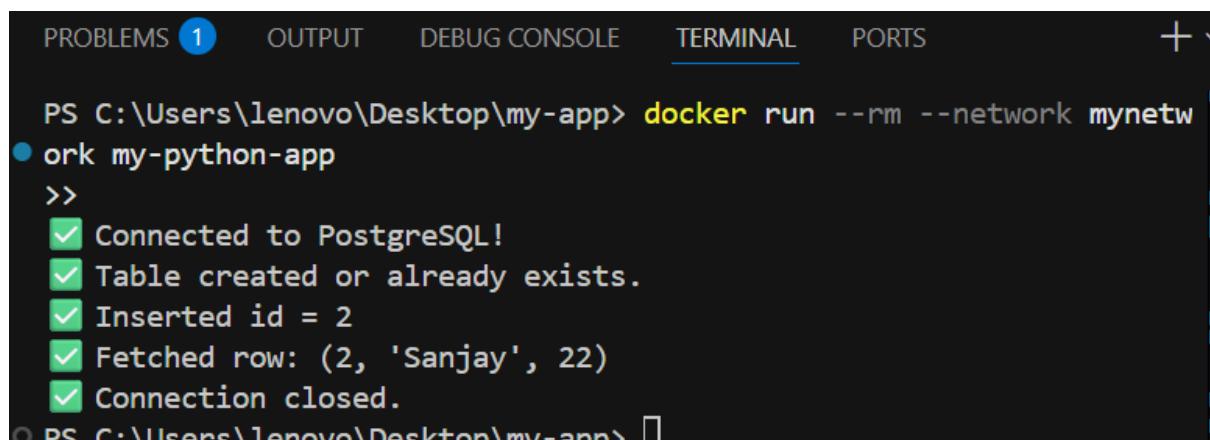
15. Rebuilding and Running the Connected App

Rebuild:

```
docker build -t my-python-app .
```

Run:

```
docker run --rm --network mynetwork my-python-app
```

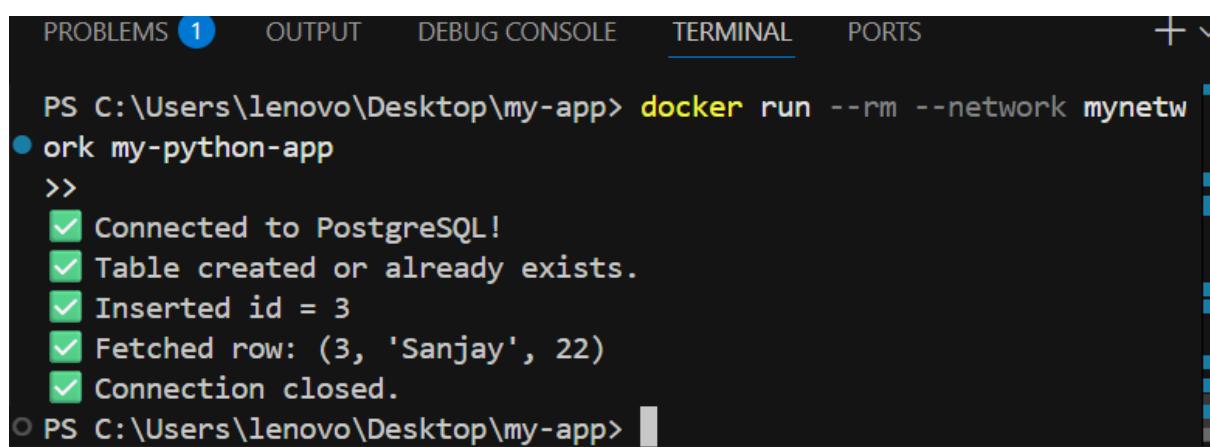


A screenshot of the VS Code interface showing the Terminal tab. The terminal window displays a PowerShell session (PS) running on a Windows machine. The command `docker run --rm --network mynetwork my-python-app` is executed, followed by a series of green checkmarks indicating successful database operations: "Connected to PostgreSQL!", "Table created or already exists.", "Inserted id = 2", "Fetched row: (2, 'Sanjay', 22)", and "Connection closed." The terminal prompt then changes to PS C:\Users\lenovo\Desktop\my-app>.

16. Final Beginner Task – Full Database Operations

Update app.py to:

- Create a table
- Insert a row
- Read data
- Print the result



A screenshot of the VS Code interface showing the Terminal tab. The terminal window displays a PowerShell session (PS) running on a Windows machine. The command `docker run --rm --network mynetwork my-python-app` is executed, followed by a series of green checkmarks indicating successful database operations: "Connected to PostgreSQL!", "Table created or already exists.", "Inserted id = 3", "Fetched row: (3, 'Sanjay', 22)", and "Connection closed." The terminal prompt then changes to PS C:\Users\lenovo\Desktop\my-app>.

17. Conclusion

This project successfully demonstrates:

- ✓ Pulling images from Docker Hub
- ✓ Creating Dockerfiles
- ✓ Building and running custom Docker images
- ✓ Running PostgreSQL as a container
- ✓ Creating Docker networks
- ✓ Connecting two containers
- ✓ Performing SQL operations inside the Dockerized Python app