Thanks, that helps a lot! Here's what we'll do based on your responses:

Setup Plan for Ubuntu

We'll use **Docker Compose**, which is **easier for beginners** because it lets you define and run multi-container applications with a single file (`docker-compose.yml`). It simplifies networking, volumes, and container relationships.

You'll:

- Use **official Docker images** for PostgreSQL, pgAdmin, and JupyterLab
- Create a **shared Docker network** so they can talk to each other
- Enable **data persistence** with volumes
- Configure your own **PostgreSQL database name, username, and password**

Step 1: Create Your Project Folder

Open your terminal and run:

```
mkdir docker-postgres-stack
cd docker-postgres-stack
```

This will be your working directory.

Step 2: Create the `docker-compose.yml` File

Inside this directory, create a file called `docker-compose.yml`:

```
touch docker-compose.yml
```

Now open it in your preferred text editor (e.g., VS Code or nano):

```
nano docker-compose.yml
```

Paste the following into the file (modify where indicated):

```
version: '3.8'

services:
  postgres:
    image: postgres:latest
    container_name: postgres
    restart: always
    environment:
      POSTGRES_USER: myuser          # <-- change this
      POSTGRES_PASSWORD: mypassword # <-- change this
      POSTGRES_DB: mydatabase        # <-- change this
    volumes:
      - pgdata:/var/lib/postgresql/data
    networks:
      - localnet

  pgadmin:
    image: dpage/pgadmin4:latest
```

```
    container_name: pgadmin
    restart: always
    environment:
      PGADMIN_DEFAULT_EMAIL: admin@admin.com  # <-- login email
      PGADMIN_DEFAULT_PASSWORD: admin         # <-- login password
    ports:
      - "8080:80"
    networks:
      - localnet

  jupyterlab:
    image: jupyter/base-notebook:latest
    container_name: jupyterlab
    restart: always
    ports:
      - "8888:8888"
    volumes:
      - ./notebooks:/home/jovyan/work
    networks:
      - localnet

volumes:
  pgdata:

networks:
  localnet:
```

Save and exit (`Ctrl + X`, then `Y`, then `Enter` in `nano`).

Step 3: Start Everything

Run this command in the same directory:

```
docker-compose up -d
```

This will:

- Download the latest PostgreSQL, pgAdmin, and JupyterLab images
- Create and start containers
- Persist data in a volume
- Map ports so you can access pgAdmin at `http://localhost:8080` and JupyterLab at `http://localhost:8888`

Step 4: Access pgAdmin and Set Up Server Connection

1. Go to `http://localhost:8080` in your browser.
2. Login using the email and password you provided in the `docker-compose.yml`.
3. Click "Add New Server".
4. Under **General**:
   - Name: `PostgresServer`
5. Under **Connection**:

- Host: `postgres` *(this is the service name, not `localhost`)*
- Port: `5432`
- Username: `myuser` *(use the one you defined)*
- Password: `mypassword`

Click **Save**. You're connected!

Step 5: Access JupyterLab and Query PostgreSQL

1. Go to `http://localhost:8888`
2. Copy the token URL from your terminal if prompted, or use passwordless login
3. Open a new notebook and run:

```
import psycopg2

conn = psycopg2.connect(
    host="postgres",
    port=5432,
    dbname="mydatabase",
    user="myuser",
    password="mypassword"
)

cursor = conn.cursor()
cursor.execute("SELECT version();")
print(cursor.fetchone())
conn.close()
```

If you see the PostgreSQL version printed, it worked!

Would you like to now:

- Set up the Jupyter notebook with useful libraries (e.g. `psycopg2`, `sqlalchemy`, `pandas`)?
- Learn how to safely stop and restart everything?
- Learn how to back up your database or use a custom SQL script on startup?

Let me know what you'd like next!

Absolutely — and no worries at all! You can easily redo everything. Here's how to **reset and relaunch your Docker stack** with your updated database name, username, and password.

Step 1: Stop and Remove Everything

In your project folder (`docker-postgres-stack`), run:

```
docker-compose down -v
```

This will:

- **Stop and remove** all running containers
- **Remove the named volume** (`pgdata`) so your old PostgreSQL data is deleted

---

Step 2: Edit the `docker-compose.yml` File

Open the file again:

```
nano docker-compose.yml
```

Update these lines under the `postgres` service to use your preferred values:

```
POSTGRES_USER: youruser
POSTGRES_PASSWORD: yourpassword
POSTGRES_DB: yourdatabase
```

Also update the pgAdmin login credentials if you want:

```
PGADMIN_DEFAULT_EMAIL: your@email.com
PGADMIN_DEFAULT_PASSWORD: youradminpassword
```

Press `Ctrl+X`, then `Y`, then `Enter` to save and exit.

Step 3: Relaunch the Stack

```
docker-compose up -d
```

Give it a few seconds, then check:

- **pgAdmin**: http://localhost:8080
- **JupyterLab**: http://localhost:8888

Use your new credentials.

Would you like me to help you:

- Save these credentials in a `.env` file (so they're not hardcoded)?
- Set up Jupyter to connect automatically to your database?
- Back up the database in case you rebuild it in the future?

Let me know what you'd like to do next!

Excellent choice! Using a `.env` file keeps your credentials cleanly separated from your `docker-compose.yml` file — which is **good practice** for security and reusability.

Step-by-Step: Move Credentials to `.env` File

Step 1: Create a `.env` File

In the same folder as your `docker-compose.yml`, run:

```
touch .env
nano .env
```

Paste this into the `.env` file — replace with your actual values:

```
# PostgreSQL
POSTGRES_USER=youruser
POSTGRES_PASSWORD=yourpassword
POSTGRES_DB=yourdatabase
```

```
# pgAdmin
PGADMIN_DEFAULT_EMAIL=youremail@example.com
PGADMIN_DEFAULT_PASSWORD=youradminpassword
```

Press `Ctrl + X`, then `Y`, then `Enter` to save.

Step 2: Update `docker-compose.yml` to Use the Variables

Open the `docker-compose.yml` file:

```
nano docker-compose.yml
```

Replace the hardcoded values with environment variable references like this:

```yaml
version: '3.8'

services:
  postgres:
    image: postgres:latest
    container_name: postgres
    restart: always
    environment:
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
      POSTGRES_DB: ${POSTGRES_DB}
    volumes:
      - pgdata:/var/lib/postgresql/data
    networks:
      - localnet

  pgadmin:
    image: dpage/pgadmin4:latest
    container_name: pgadmin
    restart: always
    environment:
      PGADMIN_DEFAULT_EMAIL: ${PGADMIN_DEFAULT_EMAIL}
      PGADMIN_DEFAULT_PASSWORD: ${PGADMIN_DEFAULT_PASSWORD}
    ports:
      - "8080:80"
    networks:
      - localnet

  jupyterlab:
    image: jupyter/base-notebook:latest
    container_name: jupyterlab
    restart: always
    ports:
      - "8888:8888"
    volumes:
      - ./notebooks:/home/jovyan/work
    networks:
      - localnet

volumes:
  pgdata:
```

```
networks:
  localnet:
```

Save and exit.

Step 3: Relaunch Everything

Use:

```
docker-compose down -v
docker-compose up -d
```

Docker Compose automatically loads variables from `.env`.

Test it

- pgAdmin login: use the email and password from `.env`
- Connect to PostgreSQL using the username, password, and database name you set

---

Optional: Ignore the `.env` File in Git

If you're using Git, create or update `.gitignore`:

```
echo ".env" >> .gitignore
```

This prevents the `.env` file (with passwords) from being pushed to a repository.

Would you like to:

- Automatically pre-load a SQL schema or data file into PostgreSQL?
- Connect from Jupyter using the `.env` file for credentials?
- Add PostgreSQL client tools (like `psql`) into Jupyter?

Let me know what's next!

Awesome — building your own **custom JupyterLab Docker image** will solve both your slow startup and let you permanently include all your tools like `psql`, `ydata-profiling`, and data science libraries.

What You'll Do

1. Create a custom `Dockerfile` based on the Jupyter base-notebook image
2. Install:
   - PostgreSQL CLI tools (`psql`)
   - All your required Python packages
3. Update `docker-compose.yml` to use your custom image

Step-by-Step Guide

Step 1: Create a `Dockerfile` in Your Project Folder

```
cd docker-postgres-stack
touch Dockerfile.jupyterlab
nano Dockerfile.jupyterlab
```

Paste this into the file:

```
FROM jupyter/base-notebook:latest

# Switch to root to install system-level packages
USER root

# Install PostgreSQL client
RUN apt-get update && \
    apt-get install -y postgresql-client && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*

# Switch back to the default Jupyter user
USER $NB_UID

# Install Python packages
RUN pip install --no-cache-dir \
    pandas \
    numpy \
    scipy \
    seaborn \
    matplotlib \
    scikit-learn \
    ydata-profiling
```

Step 2: Update `docker-compose.yml`

Change your `jupyterlab` service to **build from this Dockerfile**:

```
  jupyterlab:
    build:
      context: .
      dockerfile: Dockerfile.jupyterlab
    image: custom-jupyterlab:latest
    container_name: jupyterlab
    restart: always
    ports:
      - "8888:8888"
    volumes:
      - ./notebooks:/home/jovyan
    networks:
      - localnet
```

You no longer need the `command:` line with `pip install` or `apt install` since it's now baked into the image.

This speeds up container startup dramatically.

Step 3: Rebuild and Run

Run the following to build your image and spin up all services:

```
docker-compose down
docker-compose up --build -d
```

This will:

- Build `custom-jupyterlab` from your Dockerfile
- Start it with all tools preinstalled
- Persist your notebooks to the host via `./notebooks`

Step 4: Test It

1. Open JupyterLab: http://localhost:8888
2. Open a terminal and test `psql`:

```
psql --version
```

3. In a notebook, test:

```
import pandas as pd
import seaborn as sns
from ydata_profiling import ProfileReport
```

Would you like me to:

- Add support for `.env` variables inside Jupyter so you can use them in Python?
- Preload a notebook that connects to PostgreSQL and loads a table?

Let me know what else you'd like in the image or startup!

Great follow-up!

What Does "Install PostgreSQL Client Tools (like `psql`) in Jupyter" Mean?

When you're working in a Jupyter notebook, you're typically using Python to connect to PostgreSQL (e.g., using `psycopg2` or `sqlalchemy`). That works well.

But sometimes you may want to:

- Use `psql`, the **PostgreSQL command-line tool**, inside the JupyterLab terminal to directly run SQL queries.
- Run quick SQL commands without writing Python code.
- Troubleshoot connection issues or inspect the database schema from the terminal.

Example Use Case in JupyterLab Terminal

Let's say you open a terminal in JupyterLab and run:

```
psql -h postgres -U youruser -d yourdatabase
```

You'd get an interactive SQL prompt like this:

```
yourdatabase=>
```

Now you can run SQL directly:

```
SELECT * FROM your_table LIMIT 5;
```

Why It's Useful

- Great for **manual exploration**
- Great for **admin tasks**
- Gives you **direct access to PostgreSQL**, just like being on the database server

---

How to Enable This in Your JupyterLab Container

You just need to install the PostgreSQL client (`postgresql-client`) inside the JupyterLab container.

Easiest Way (Add to your Docker Compose command): Update the `jupyterlab` service in your `docker-compose.yml` like this:

```
command: bash -c "apt-get update && apt-get install -y postgresql-client && pip install -r /tmp/requirem
```

This will:

1. Install `psql` via `apt-get`
2. Install your Python libraries
3. Start JupyterLab

Then From JupyterLab:

1. Go to the **JupyterLab interface** at http://localhost:8888
2. Click on **"Terminal"**
3. Run:

```
psql -h postgres -U youruser -d yourdatabase
```

You'll be prompted for your password (use the one from your `.env` file).

Would you like me to:

- Modify your `docker-compose.yml` for you?
- Set it up so the password is passed automatically (to avoid typing it each time)?
- Add a helper script or notebook to open the connection via Python and SQL?

Let me know what feels useful!