Yes, you can absolutely manage this entire self-hosted stack with a single Docker Compose file. This is the ideal approach as it defines all of your services, networks, and volumes in one place, making your application reproducible and easy to manage.
Here is the complete configuration to run the database, the NLP model, the dashboard, and the scheduled scraper script.

The docker-compose.yml File

This is the central file that orchestrates everything. It defines four main services:

1. **db**: The PostgreSQL database.
2. **ollama**: The local LLM for NLP tasks.
3. **app**: The Python Dash dashboard, served with Gunicorn.
4. **scheduler**: A dedicated container that runs the scraping script on a cron schedule.

<!-- end list -->

```
# docker-compose.yml

version: '3.8'

services:
  # 1. PostgreSQL Database Service
  db:
    image: postgres:16
    container_name: its_db
    restart: always
    environment:
      - POSTGRES_USER=${DB_USER}
      - POSTGRES_PASSWORD=${DB_PASSWORD}
      - POSTGRES_DB=${DB_NAME}
    volumes:
      - postgres_data:/var/lib/postgresql/data # Persists database
data
    ports:
      - "5432:5432" # Expose port for local connections if needed

  # 2. Local LLM Service (Ollama)
  ollama:
    image: ollama/ollama
    container_name: its_ollama
    restart: always
    volumes:
      - ollama_data:/root/.ollama # Persists downloaded models
    ports:
      - "11434:11434" # Expose Ollama API port

  # 3. Dash Web Application Service
  app:
    container_name: its_dashboard
    restart: always
```

```
    build:
      context: .
      dockerfile: Dockerfile # Uses the main Dockerfile
    ports:
      - "8050:8050" # Expose dashboard port to your machine
    environment:
      -
DATABASE_URL=postgresql://${DB_USER}:${DB_PASSWORD}@db:5432/${DB_NAME}
      - OLLAMA_URL=http://ollama:11434
    depends_on:
      - db
      - ollama
    command: gunicorn --bind 0.0.0.0:8050 --workers 4 app:server #
Starts the app with Gunicorn

  # 4. Scheduled Scraper Service
  scheduler:
    container_name: its_scheduler
    restart: always
    build:
      context: .
      dockerfile: Dockerfile # Re-uses the app's Docker image for
efficiency
    environment:
      -
DATABASE_URL=postgresql://${DB_USER}:${DB_PASSWORD}@db:5432/${DB_NAME}
      - OLLAMA_URL=http://ollama:11434
    depends_on:
      - db
      - ollama
    command: cron -f # Runs the cron daemon in the foreground

volumes:
  postgres_data:
  ollama_data:
```

Supporting Files

You'll need to create the following files in the same directory as your docker-compose.yml.

1. Dockerfile

This file defines how to build the Python environment for both the **app** and the **scheduler**.

```
# Dockerfile

# Use an official Python runtime as a parent image
FROM python:3.10-slim
```

```
# Set the working directory in the container
WORKDIR /app

# Install cron for the scheduler service
RUN apt-get update && apt-get install -y cron

# Copy the requirements file into the container
COPY requirements.txt .

# Install any needed packages specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Copy the cron job file to the cron directory
COPY crontab /etc/cron.d/scraper-cron
RUN chmod 0644 /etc/cron.d/scraper-cron

# Create the cron job
RUN crontab /etc/cron.d/scraper-cron

# Copy the rest of the application's code
COPY . .

# The app service will override this with its own command
CMD ["cron", "-f"]
```

2. requirements.txt

A list of the core Python libraries your application will need.

```
# requirements.txt
dash
gunicorn
pandas
psycopg2-binary
selenium
beautifulsoup4
xgboost
scikit-learn
requests
```

3. crontab

This file defines the schedule for your scraping task. This example runs the script every day at 2:00 AM.

```
# crontab
# Runs the scraper script every day at 2:00 AM
```

```
0 2 * * * python /app/main_pipeline.py >> /var/log/cron.log 2>&1
# An empty line is required at the end of this file for cron to work
correctly
```

4. .env File

This file stores your secrets. Docker Compose will automatically load these as environment variables. **Do not commit this file to version control.**

```
# .env
# PostgreSQL Credentials
DB_USER=admin
DB_PASSWORD=your_super_secret_password
DB_NAME=its_data
```

How to Use It

1. **Create the Files**: Place all the files (docker-compose.yml, Dockerfile, requirements.txt, crontab, .env) in a single project folder. You will also have your Python application code (e.g., app.py, main_pipeline.py) in this folder.
2. **Build and Run**: Open a terminal in the project folder and run the following command:
   `docker-compose up --build -d`
   This will build your Python image, download the Postgres and Ollama images, and start all four containers in the background (-d).
3. **Pull an LLM Model**: The first time you run this, you need to tell Ollama which model to download.
   `docker-compose exec ollama ollama pull llama3`

4. **Access Your Dashboard**: Open your web browser and navigate to http://localhost:8050.

Your entire platform is now running locally, orchestrated by a single, clean docker-compose.yml file.