

AI-03596 Fix n8n Docker Setup

👤 Owner	© Rigel Arcayan
☰ Tags	Project
🕒 Created time	June 20, 2026 8:20 AM

Explain the following **in detail**:

```
docker volume create n8n_data
```

```
docker run -it --rm \  
  --name n8n \  
  -p 5678:5678 \  
  -e GENERIC_TIMEZONE="<YOUR_TIMEZONE>" \  
  -e TZ="<YOUR_TIMEZONE>" \  
  -e N8N_ENFORCE_SETTINGS_FILE_PERMISSIONS=true \  
  -e N8N_RUNNERS_ENABLED=true \  
  -e DB_TYPE=postgresdb \  
  -e DB_POSTGRESDB_DATABASE=<POSTGRES_DATABASE> \  
  -e DB_POSTGRESDB_HOST=<POSTGRES_HOST> \  
  -e DB_POSTGRESDB_PORT=<POSTGRES_PORT> \  
  -e DB_POSTGRESDB_USER=<POSTGRES_USER> \  
  -e DB_POSTGRESDB_SCHEMA=<POSTGRES_SCHEMA> \  
  -e DB_POSTGRESDB_PASSWORD=<POSTGRES_PASSWORD> \  
  -v n8n_data:/home/node/.n8n \  
  docker.n8n.io/n8nio/n8n
```

This sequence of commands is used to spin up **n8n** (a powerful, node-based workflow automation tool) inside a Docker container, configuring it to persist its data and connect to an external PostgreSQL database instead of its default SQLite

database.

Here is a detailed breakdown of exactly what each line does.

Part 1: Creating the Volume

```
docker volume create n8n_data
```

- **docker volume create:** Tells Docker to create a dedicated, managed storage space on your host machine's disk.
- **n8n_data:** The name given to this specific volume.
- **Why it matters:** Docker containers are ephemeral by default—if the container is deleted, all data inside it vanishes. This command prepares a safe, permanent storage area for n8n's internal configurations, encryption keys, and local files.

Part 2: Running the Container

```
docker run -it --rm \
```

- **docker run:** Instructs Docker to create and start a new container from a specified image.
- **it:** Combines two flags: `-i` (interactive) keeps the container's standard input open, and `-t` (tty) allocates a pseudo-TTY. Together, they allow you to see the real-time logs in your terminal and interact with the process if needed.
- **-rm:** Automatically deletes the container when it exits or stops. This is great for keeping your Docker environment clean, especially during testing.

```
--name n8n \  
-p 5678:5678 \  
n8n
```

- **-name n8n:** Assigns the friendly name n8n to the running container so you can easily reference it later (e.g., `docker stop n8n`) instead of relying on a random, auto-generated name.


```
-e DB_POSTGRESDB_HOST=<POSTGRES_HOST> \  
-e DB_POSTGRESDB_PORT=<POSTGRES_PORT> \  
-e DB_POSTGRESDB_USER=<POSTGRES_USER> \  
-e DB_POSTGRESDB_SCHEMA=<POSTGRES_SCHEMA> \  
-e DB_POSTGRESDB_PASSWORD=<POSTGRES_PASSWORD> \
```

- **DB_TYPE=postgresdb:** Tells n8n to switch its backend database engine to PostgreSQL.
- The remaining lines are placeholders (indicated by < >) where you must input your specific database connection details: the database name, host IP address/domain, port (usually 5432), username, schema (usually public), and password.

Volume Mounting and the Image

```
-v n8n_data:/home/node/.n8n \
```

- **v:** Mounts a volume into the container using the syntax `volume_name:container_path`.
- This mounts the `n8n_data` volume created in Part 1 directly to `/home/node/.n8n` inside the container. This specific folder is where n8n stores crucial assets like your encryption keys (used to secure your API credentials). Even if you stop and delete this container, your data remains safely stored in the volume and will reload the next time you run this command.

```
docker.n8n.io/n8nio/n8n
```

- This is the official target **Docker image** that you are downloading and running. It pulls the latest stable version of n8n directly from n8n's official image registry.

Convert to a docker-compose file

```
docker volume create n8n_data
```

```
docker run -it --rm \  
--name n8n \  
--name n8n \  
--name n8n \
```

```
-p 5678:5678 \  
-e GENERIC_TIMEZONE="<YOUR_TIMEZONE>" \  
-e TZ="<YOUR_TIMEZONE>" \  
-e N8N_ENFORCE_SETTINGS_FILE_PERMISSIONS=true \  
-e N8N_RUNNERS_ENABLED=true \  
-e DB_TYPE=postgresdb \  
-e DB_POSTGRESDB_DATABASE=<POSTGRES_DATABASE> \  
-e DB_POSTGRESDB_HOST=<POSTGRES_HOST> \  
-e DB_POSTGRESDB_PORT=<POSTGRES_PORT> \  
-e DB_POSTGRESDB_USER=<POSTGRES_USER> \  
-e DB_POSTGRESDB_SCHEMA=<POSTGRES_SCHEMA> \  
-e DB_POSTGRESDB_PASSWORD=<POSTGRES_PASSWORD> \  
-v n8n_data:/home/node/.n8n \  
docker.n8n.io/n8nio/n8n
```

Here is the equivalent docker-compose.yml file for your docker run command.

```
version: '3.8'  
  
services:  
  n8n:  
    image: docker.n8n.io/n8nio/n8n  
    container_name: n8n  
    restart: unless-stopped  
    ports:  
      - "5678:5678"  
    environment:  
      - GENERIC_TIMEZONE=<YOUR_TIMEZONE>  
      - TZ=<YOUR_TIMEZONE>  
      - N8N_ENFORCE_SETTINGS_FILE_PERMISSIONS=true  
      - N8N_RUNNERS_ENABLED=true  
      - DB_TYPE=postgresdb  
      - DB_POSTGRESDB_DATABASE=<POSTGRES_DATABASE>  
      - DB_POSTGRESDB_HOST=<POSTGRES_HOST>  
      - DB_POSTGRESDB_PORT=<POSTGRES_PORT>
```

```
- DB_POSTGRESDB_USER=<POSTGRES_USER>
- DB_POSTGRESDB_SCHEMA=<POSTGRES_SCHEMA>
- DB_POSTGRESDB_PASSWORD=<POSTGRES_PASSWORD>
volumes:
  - n8n_data:/home/node/.n8n
```

```
volumes:
  n8n_data:
    external: false
```

A Few Quick Tips:

- **restart: unless-stopped:** I added this policy so your n8n instance automatically boots back up if your Docker daemon or server restarts.
- **Placeholders:** Don't forget to swap out the <YOUR_TIMEZONE>, <POSTGRES_HOST>, etc., with your actual database and location details before running it.
- **How to run:** Save this as docker-compose.yml in your desired directory and launch it in the background using:

```
docker compose up -d
```

Convert to compose file but use Postgres-Shared db, user-posgres, password-SecurePassword1, domain-blindcaveman.duckdns.org

```
docker volume create n8n_data
```

```
docker run -it --rm \
--name n8n \
-p 5678:5678 \
-e GENERIC_TIMEZONE="<YOUR_TIMEZONE>" \
-e TZ="<YOUR_TIMEZONE>" \
-e N8N_ENFORCE_SETTINGS_FILE_PERMISSIONS=true \
-e N8N_RUNNERS_ENABLED=true \
-e DB_TYPE=postgresdb \
-e DB_POSTGRESDB_DATABASE=<POSTGRES_DATABASE> \
```

```
-e DB_POSTGRESDB_HOST=<POSTGRES_HOST> \  
-e DB_POSTGRESDB_PORT=<POSTGRES_PORT> \  
-e DB_POSTGRESDB_USER=<POSTGRES_USER> \  
-e DB_POSTGRESDB_SCHEMA=<POSTGRES_SCHEMA> \  
-e DB_POSTGRESDB_PASSWORD=<POSTGRES_PASSWORD> \  
-v n8n_data:/home/node/.n8n \  
docker.n8n.io/n8nio/n8n
```

Here is the updated `docker-compose.yml` file.

I have bundled both the **n8n** service and a **PostgreSQL** service together so they can communicate seamlessly. I also included the specific domain and database credentials you requested, and added the `WEBHOOK_URL` environment variable, which n8n requires to function properly when running behind a custom domain.

```
version: '3.8'  
  
services:  
  postgres:  
    image: postgres:16-alpine  
    container_name: n8n_postgres  
    restart: unless-stopped  
    environment:  
      - POSTGRES_USER=postgres  
      - POSTGRES_PASSWORD=SecurePassword1  
      - POSTGRES_DB=postgres  
    volumes:  
      - db_data:/var/lib/postgresql/data  
    healthcheck:  
      test: ["CMD-SHELL", "pg_isready -U postgres"]  
      interval: 5s  
      timeout: 5s  
      retries: 5  
  
  n8n:  
    image: docker.n8n.io/n8nio/n8n
```

```

container_name: n8n
restart: unless-stopped
ports:
  - "5678:5678"
environment:
  - GENERIC_TIMEZONE=America/New_York # <-- Change to your actual timezone
  - TZ=America/New_York # <-- Change to your actual timezone
  - N8N_ENFORCE_SETTINGS_FILE_PERMISSIONS=true
  - N8N_RUNNERS_ENABLED=true

# Database Configuration
  - DB_TYPE=postgresdb
  - DB_POSTGRESDB_HOST=postgres
  - DB_POSTGRESDB_PORT=5432
  - DB_POSTGRESDB_DATABASE=postgres
  - DB_POSTGRESDB_USER=postgres
  - DB_POSTGRESDB_PASSWORD=SecurePassword1
  - DB_POSTGRESDB_SCHEMA=public

# Domain Configuration
  - WEBHOOK_URL=https://blindcaveman.duckdns.org/
volumes:
  - n8n_data:/home/node/.n8n
depends_on:
  postgres:
    condition: service_healthy

volumes:
  n8n_data:
  db_data:

```

Key Improvements Made:

- **depends_on with Healthcheck:** n8n will wait to boot up until the PostgreSQL database is fully ready to accept connections, preventing initial connection errors.
- **WEBHOOK_URL:** Set to your DuckDNS domain. This ensures that any webhooks you create inside n8n use your actual external URL instead of a `localhost` address.
- **Network Sharing:** Because both services live in the same Compose file, they share a default network. n8n can securely talk to the database using `DB_POSTGRESDB_HOST=postgres` without exposing port `5432` to the public internet.

(Note: Don't forget to double-check the `TZ` and `GENERIC_TIMEZONE` variables to match your local runtime environment!)

I mean separate the db or remove the db service because I have a working db in my dockerlab-net (use this bridge network)

Got it! You already have a shared Postgres instance running on an existing user-defined network called `dockerlab-net`.

Here is the cleaned-up `docker-compose.yml` that removes the internal database service, connects n8n to your existing `dockerlab-net` bridge network, and targets your specific database credentials.

```
version: '3.8'

services:
  n8n:
    image: docker.n8n.io/n8nio/n8n
    container_name: n8n
    restart: unless-stopped
    ports:
      - "5678:5678"
    environment:
      - GENERIC_TIMEZONE=America/New_York # <-- Update to your
        r timezone
      - TZ=America/New_York           # <-- Update to your time
        zone
```

```

- N8N_ENFORCE_SETTINGS_FILE_PERMISSIONS=true
- N8N_RUNNERS_ENABLED=true

# Database Configuration (Connecting to your external P
ostgres)
- DB_TYPE=postgresdb
- DB_POSTGRESDB_HOST=Postgres-Shared
- DB_POSTGRESDB_PORT=5432
- DB_POSTGRESDB_DATABASE=postgres
- DB_POSTGRESDB_USER=postgres
- DB_POSTGRESDB_PASSWORD=SecurePassword1
- DB_POSTGRESDB_SCHEMA=public

# Domain Configuration
- WEBHOOK_URL=https://blindcaveman.duckdns.org/
volumes:
- n8n_data:/home/node/.n8n
networks:
- dockerlab-net

volumes:
  n8n_data:

networks:
  dockerlab-net:
    external: true

```

What Changed?

- **networks block:** Added the `dockerlab-net` network to the n8n service and declared it as `external: true` at the bottom. This tells Docker Compose not to create a new network, but to hook into the one you already have.
- **DB_POSTGRESDB_HOST:** Set to `Postgres-Shared`. As long as your existing database container is named `Postgres-Shared` and is attached to `dockerlab-net`, Docker's internal DNS will route the traffic flawlessly.

- **Database Cleaned:** Removed the local Postgres service definition entirely.

What do I need to do in pgadmin to setup the n8n db prior to executing the compose file?

You actually don't have to do much! n8n automatically runs database migrations on its first startup, meaning **it will create all the necessary tables, columns, and schemas on its own**. However, because you are using a shared Postgres instance (`Postgres-Shared`), you just need to ensure the target database physically exists and the user has the right permissions.

Based on your previous setup (`DB_POSTGRESDB_DATABASE=postgres`), the `postgres` database already exists by default in any PostgreSQL instance. If you want to use that default database, you don't need to do anything in pgAdmin.

If you prefer to isolate n8n into its own dedicated database (highly recommended for a shared DB instance), follow these quick steps in pgAdmin before firing up the compose file:

1. Create a Dedicated Database (Recommended)

1. Open **pgAdmin** and connect to your `Postgres-Shared` server.
2. Right-click on **Databases** → **Create** → **Database...**
3. Name it `n8n` (or whatever you prefer).
4. Click **Save**.

2. Check User Permissions

Ensure that the user `postgres` has full privileges on this new database (by default, the superuser `postgres` has permission to do everything, so you should be completely fine here).

Update Your Compose File (If you changed the DB name)

If you created a new database named `n8n` in step 1, just tweak that single line in your `docker-compose.yml` environment block before running `docker compose up -d`:

Database Configuration

```
- DB_TYPE=postgresdb
- DB_POSTGRESDB_HOST=Postgres-Shared
- DB_POSTGRESDB_PORT=5432
- DB_POSTGRESDB_DATABASE=n8n           # <-- Change from 'postgres' to 'n8n'
- DB_POSTGRESDB_USER=postgres
```

```
- DB_POSTGRESDB_PASSWORD=SecurePassword1
- DB_POSTGRESDB_SCHEMA=public
```

Once you deploy the compose file, you can watch the n8n container logs (`docker logs -f n8n`), and you'll see it instantly start writing its initial framework tables into your Postgres instance!

How to configure the NPM Proxy

To get Nginx Proxy Manager (NPM) routing traffic correctly to your n8n container through your `dockerlab-net` network, follow these configuration steps:

1. Ensure NPM is on the same network

For NPM to see n8n using its container name (`n8n`), NPM must also be attached to your `dockerlab-net` network.

- If you deployed NPM via a different docker-compose file, make sure it has the exact same network block mapping:

YAML

```
networks:
  dockerlab-net:
    external: true
```

2. Configure the Proxy Host in the NPM Web UI

1. Open your NPM dashboard (usually port `81`) and navigate to **Hosts > Proxy Hosts > Add Proxy Host**.
2. Set up the **Details** tab exactly like this:
 - **Domain Names:** `blindcaveman.duckdns.org`
 - **Scheme:** `http`
 - **Forward Hostname / IP:** `n8n` (Since they share the `dockerlab-net` network, Docker's DNS maps the container name directly).
 - **Forward Port:** `5678`

- **Block Common Exploits:** *Enabled* (Recommended)
- **Websockets Support: ENABLED** (*CRITICAL: n8n relies on WebSockets for its UI updates. If you leave this turned off, the execution screen will lag or throw connection errors*).

3. Configure the SSL Tab

1. Switch to the **SSL** tab at the top.
2. Select **Request a new SSL Certificate** from the dropdown.
3. Turn on **Force SSL** and **HTTP/2 Support**.
4. Enter your email address to agree to the Let's Encrypt Terms of Service.
5. Click **Save**.

Why this works seamlessly

Because both n8n and NPM share `dockerlab-net`, NPM can talk directly to `http://n8n:5678` behind the scenes. You don't even technically need to expose `"5678:5678"` to the public in your n8n docker-compose file anymore, because NPM handles all external entries securely through port `443`.