My Experience Setting Up n8n on Windows 11 Pro Using Docker

Setting up n8n on my Windows 11 Pro machine was a journey filled with some unexpected challenges, but I managed to overcome them step by step. Here's how it went:

Step 1: Application Not Running After Installation

After downloading the n8n application and attempting to run it on my Windows 11 Pro system, the app failed to start without providing any explanation. I was initially unsure of what the issue was.

What I Did

1. I double-checked that **Hyper-V** was enabled on my system:
    - Opened Turn Windows features on or off and confirmed that **Hyper-V** was checked.
2. After some digging, I discovered that the issue was with an outdated version of the **Windows Subsystem for Linux (WSL)** on my machine.

Solution

To fix this, I updated WSL via PowerShell using the following command:

wsl --update

After running this command, the Docker Desktop application worked as expected, and I was ready to proceed with setting up n8n.

Step 2: Running n8n in Docker

With Docker Desktop up and running, I moved on to setting up n8n as a Docker container. Here's what I did:

Steps to Set Up n8n

1. I pulled the latest n8n Docker image:
2. docker pull n8nio/n8n
3. Then, I created and started the container:
4. docker run -d -p 5678:5678 --name n8n-container n8nio/n8n
    - **Flags I used:**
        - -d: Run the container in detached mode.
        - -p 5678:5678: Map the container port 5678 to the host port 5678.
        - --name: Assigned a specific name to the container for easier reference.

5. To ensure everything was working, I verified the container status:
6. docker ps

The output confirmed that the container was running and listening on port 5678.

Checking Logs

I also checked the container logs to confirm that n8n had initialized properly:

docker logs n8n-container

The logs indicated:

- n8n was ready on http://0.0.0.0:5678.
- There was a warning about permissions related to /home/node/.n8n/config, but it didn't stop the application from running.

Step 3: Resolving Connection Issues

Despite n8n being up and running, I couldn't connect to it via http://localhost:5678 in my browser. I encountered a ERR_CONNECTION_REFUSED error.

Firewall Configuration

To fix this, I suspected that the issue might be related to Windows Firewall blocking the port.

1. I opened **Windows Defender Firewall** and created a new inbound rule:
    - Rule Type: **Port**
    - Protocol: **TCP**
    - Specific Local Port: 5678
    - Action: **Allow the connection**
    - Applied to: **Domain**, **Private**, and **Public** networks.
2. After applying the rule, I tested the connection again, and this time, I was able to access n8n at http://localhost:5678.

Step 4: Resolving Port Mapping Issue

One key issue I encountered was related to incorrect port mapping when initially setting up the container. Here's a breakdown:

What Was Incorrectly Set?

- **Port Mapping**:

- In the initial setup (docker ps), the host port (32772) was randomly assigned instead of explicitly mapped to 5678. This made it inaccessible via http://localhost:5678.

What Corrected the Issue?

1. **Proper Port Mapping**:
    - Recreated the container with the correct port mapping:
    - docker run -d -p 5678:5678 --name n8n-container n8nio/n8n
    - This explicitly binds the host port 5678 to the container port 5678, allowing access via http://localhost:5678.
2. **Firewall Adjustment**:
    - Configured Windows Defender Firewall to allow incoming traffic on port 5678, ensuring no external blockages.

Checking Logs

The logs showed:

- **Permission Warning**:
    - "Permissions 0644 for n8n settings file /home/node/.n8n/config are too wide. This is ignored for now, but in the future n8n will attempt to change the permissions automatically."

Next Steps to Address Warning

To resolve this warning, I will start the container with the recommended environment variable:

docker run -d -p 5678:5678 --name n8n-container -e N8N_ENFORCE_SETTINGS_FILE_PERMISSIONS=true n8nio/n8n

---

Lessons Learned

This experience taught me a lot about troubleshooting in Docker and configuring my Windows environment. Here are some key takeaways:

1. **Keep WSL Updated**: Ensuring WSL is up to date is crucial for running Docker Desktop effectively on Windows.
2. **Verify Port Mapping**: Always double-check the port mapping when running containers to avoid connectivity issues.
3. **Firewall Rules Matter**: Configuring Windows Firewall is essential when exposing ports for Docker containers.

Next Steps

Now that n8n is up and running, my next step is to resolve the file permissions warning in the logs by setting the environment variable:

N8N_ENFORCE_SETTINGS_FILE_PERMISSIONS=true

With this resolved, I'm excited to start building and automating workflows in n8n!

This journey wasn't without its challenges, but each step was an opportunity to learn more about Docker and Windows configuration. If you're setting up n8n and run into similar issues, I hope this write-up helps you navigate through them.