Using the `--link` option is deprecated and not recommended for connecting Docker containers. Instead, you should use Docker networks to handle container-to-container communication. Docker networks provide a more robust and flexible way to manage connections between containers.

Here’s how you can connect your Odoo container to your existing PostgreSQL (`db3`) container using Docker networks:

### 1. \*\*Ensure Both Containers Are on the Same Network\*\*

Make sure that both your Odoo container and the PostgreSQL container (`db3`) are connected to the same Docker network. If you’re using an existing network like `nginx\_proxy\_manager\_default`, ensure both containers are attached to this network.

### 2. \*\*Inspect the Existing Network\*\*

You can inspect the existing Docker network to verify that it includes the `db3` container:

```bash

docker network inspect nginx\_proxy\_manager\_default

```

Look for the `db3` container in the list of connected containers.

### 3. \*\*Configure Your `docker-compose.yml` File\*\*

You should specify the network in your `docker-compose.yml` file and make sure your Odoo container uses this network to connect to the PostgreSQL container. Here’s how to configure your `docker-compose.yml` file:

```yaml

version: '3.8'

services:

odoo1:

image: odoo:17

container\_name: odoo-instance1

depends\_on:

- db3

environment:

HOST: db3

USER: odoo

PASSWORD: odoo

DB\_NAME: odoo\_db1 # Ensure this database exists in db3

ports:

- "8069:8069"

networks:

- nginx\_proxy\_manager\_default

networks:

nginx\_proxy\_manager\_default:

external: true

```

### 4. \*\*Start or Restart Your Containers\*\*

If you have updated your `docker-compose.yml` file or want to restart your containers to apply changes, use:

```bash

docker-compose up -d

```

### 5. \*\*Verify Connectivity\*\*

To ensure that the Odoo container can communicate with the PostgreSQL container:

1. \*\*Check Logs\*\*: Inspect the logs of your Odoo container to see if it starts without errors:

```bash

docker logs odoo-instance1

```

2. \*\*Connect to the Odoo Container\*\*: You can enter the Odoo container to check connectivity manually if needed:

```bash

docker exec -it odoo-instance1 bash

```

Inside the Odoo container, you can use tools like `ping` to check connectivity to the `db3` container:

```bash

ping db3

```

### Summary

- \*\*Network Configuration\*\*: Use Docker networks to connect containers. Ensure both Odoo and PostgreSQL containers are on the same network.

- \*\*Environment Variables\*\*: Configure Odoo to connect to PostgreSQL using environment variables (`HOST`, `USER`, `PASSWORD`, `DB\_NAME`).

- \*\*Container Management\*\*: Use Docker Compose to manage and deploy containers.

Using Docker networks is the modern and recommended approach for inter-container communication, offering better flexibility and support compared to the deprecated `--link` method.

If you don’t have a PostgreSQL container (`db3`) and need to add PostgreSQL as a service within the same `docker-compose.yml`, you can define it alongside your Odoo instance. Here’s how you can configure your `docker-compose.yml` file to include both Odoo and PostgreSQL, while also specifying volumes for data and custom addons.

### Example `docker-compose.yml` with PostgreSQL and Odoo

```yaml

version: '3.8'

services:

db:

image: postgres:13

container\_name: odoo-db

environment:

POSTGRES\_USER: odoo

POSTGRES\_PASSWORD: odoo

POSTGRES\_DB: odoo\_db

volumes:

- postgres-data:/var/lib/postgresql/data

networks:

- nginx\_proxy\_manager\_default

odoo:

image: odoo:17

container\_name: odoo-instance

depends\_on:

- db

environment:

HOST: db

USER: odoo

PASSWORD: odoo

DB\_NAME: odoo\_db

ports:

- "8069:8069"

networks:

- nginx\_proxy\_manager\_default

volumes:

- odoo-data:/var/lib/odoo

- odoo-extra-addons:/mnt/extra-addons

networks:

nginx\_proxy\_manager\_default:

external: true

volumes:

postgres-data:

driver: local

odoo-data:

driver: local

odoo-extra-addons:

driver: local

```

### Key Components:

1. \*\*PostgreSQL Service (`db`):\*\*

- \*\*`image: postgres:13`\*\*: Specifies the PostgreSQL Docker image.

- \*\*`environment`\*\*: Sets PostgreSQL environment variables:

- `POSTGRES\_USER`: Username for PostgreSQL.

- `POSTGRES\_PASSWORD`: Password for PostgreSQL.

- `POSTGRES\_DB`: Name of the initial database.

- \*\*`volumes`\*\*: Persists PostgreSQL data on the host system.

- \*\*`networks`\*\*: Connects the PostgreSQL container to the specified network.

2. \*\*Odoo Service (`odoo`):\*\*

- \*\*`image: odoo:17`\*\*: Specifies the Odoo Docker image.

- \*\*`depends\_on`\*\*: Ensures the `db` service starts before Odoo.

- \*\*`environment`\*\*: Configures Odoo to connect to PostgreSQL:

- `HOST`: Hostname of the PostgreSQL service.

- `USER`: PostgreSQL username.

- `PASSWORD`: PostgreSQL password.

- `DB\_NAME`: Database name.

- \*\*`ports`\*\*: Maps port 8069 on the host to port 8069 in the container.

- \*\*`networks`\*\*: Connects the Odoo container to the specified network.

- \*\*`volumes`\*\*:

- \*\*`odoo-data`\*\*: Persists Odoo data.

- \*\*`odoo-extra-addons`\*\*: Mounts a directory for custom Odoo addons.

3. \*\*Networks:\*\*

- \*\*`nginx\_proxy\_manager\_default`\*\*: Use the existing external network if required.

4. \*\*Volumes:\*\*

- \*\*`postgres-data`\*\*: Persists PostgreSQL database files.

- \*\*`odoo-data`\*\*: Persists Odoo application data.

- \*\*`odoo-extra-addons`\*\*: Mounts custom Odoo addons directory.

### Steps to Apply Configuration:

1. \*\*Create and Start Containers:\*\*

Run Docker Compose to create and start your services:

```bash

docker-compose up -d

```

2. \*\*Verify Services:\*\*

Check the status and logs of your services to ensure they are running correctly:

```bash

docker-compose ps

docker-compose logs -f

```

3. \*\*Access Odoo:\*\*

Access your Odoo instance through your browser at `http://<your-vps-ip>:8069`.

### Summary

- \*\*Combined Services\*\*: This `docker-compose.yml` file sets up both PostgreSQL and Odoo.

- \*\*Volumes\*\*: Persistent storage for both PostgreSQL and Odoo.

- \*\*Networks\*\*: Connects services using a specified network.

This setup ensures that both Odoo and PostgreSQL run in separate containers but can communicate over the same Docker network, and data is persisted between container restarts or recreations.

The error you're encountering suggests that there's a problem with the file path or permissions when attempting to upload your custom addons to the specified directory in Odoo's Docker container. Here's a step-by-step guide to address this issue:

### 1. \*\*Check Directory Path\*\*

Ensure that the directory `/home/erp/.local/share/Odoo/addons/17.0/` exists inside the Docker container. You can do this by executing a shell in the running container:

```bash

docker exec -it <container\_id> /bin/bash

```

Then, navigate to the directory and verify:

```bash

cd /home/erp/.local/share/Odoo/addons/17.0/

ls

```

If the directory doesn't exist, you might need to create it:

```bash

mkdir -p /home/erp/.local/share/Odoo/addons/17.0/

```

### 2. \*\*Check File Permissions\*\*

Ensure that you have the correct permissions to write to the directory. You can set the appropriate permissions within the Docker container:

```bash

chmod -R 755 /home/erp/.local/share/Odoo/addons/17.0/

chown -R erp:erp /home/erp/.local/share/Odoo/addons/17.0/

```

Replace `erp:erp` with the correct user and group if they are different in your Docker container setup.

### 3. \*\*Verify Docker Volume Mounts\*\*

If you are using Docker volumes to mount directories from the host machine to the container, ensure that the volume is correctly mounted and points to the correct directory. You can check the volume mappings in your `docker-compose.yml` or the `docker run` command.

For example, in `docker-compose.yml`, it might look like this:

```yaml

volumes:

- /path/on/host/addons:/home/erp/.local/share/Odoo/addons/17.0

```

Ensure that `/path/on/host/addons` on your host machine has the correct permissions and contains the necessary files.

### 4. \*\*Upload Custom Addons Correctly\*\*

When uploading files to Docker containers, it's often easier to use `docker cp` or ensure that file synchronization is set up correctly. To copy files into a running Docker container, use:

```bash

docker cp /path/on/host/addons/<your\_module> <container\_id>:/home/erp/.local/share/Odoo/addons/17.0/

```

### 5. \*\*Check for File System Issues\*\*

Ensure that there are no file system issues or disk space problems. Sometimes, errors like `NoSuchFile` can occur if the file system is corrupted or full.

### 6. \*\*Restart Odoo Service\*\*

After uploading the custom addons, restart the Odoo service inside the container to ensure that it picks up the new addons:

```bash

docker exec -it <container\_id> /bin/bash

# Inside the container

supervisorctl restart odoo

```

### 7. \*\*Inspect Logs for More Details\*\*

Check the Odoo logs for more detailed error messages. Logs can provide additional insights into what's going wrong:

```bash

docker logs <container\_id>

```

### Example Steps Summary

1. \*\*Access the Container\*\*:

```bash

docker exec -it <container\_id> /bin/bash

```

2. \*\*Verify/Create Directory\*\*:

```bash

cd /home/erp/.local/share/Odoo/addons/17.0/

mkdir -p /home/erp/.local/share/Odoo/addons/17.0/

```

3. \*\*Set Permissions\*\*:

```bash

chmod -R 755 /home/erp/.local/share/Odoo/addons/17.0/

chown -R erp:erp /home/erp/.local/share/Odoo/addons/17.0/

```

4. \*\*Upload Files\*\*:

```bash

docker cp /path/on/host/addons/<your\_module> <container\_id>:/home/erp/.local/share/Odoo/addons/17.0/

```

5. \*\*Restart Odoo Service\*\*:

```bash

docker exec -it <container\_id> /bin/bash

supervisorctl restart odoo

```

By following these steps, you should be able to resolve the issue with uploading and using custom addons in Odoo running in a Docker container.