

# Technical Incident Report: Network Conflict Resolution

**Subject:** Resolution of IP Routing Conflict between Odoo ERP and Dockerized WordPress Environment

**Date:** October 26, 2023

**System Environment:** Ubuntu 20.04 (Linux)

**Priority Level:** Critical (Business Continuity)

## 1. Executive Summary

This document details the technical conflict identified between the corporate Odoo ERP access requirements and a local Dockerized WordPress development environment.

While the WordPress environment was successfully stabilized regarding internal memory and plugin resource usage (as detailed in previous logs), its deployment via Docker created a **Subnet Collision**. This collision routed traffic intended for the corporate Odoo server (Work) into the local Docker virtual network (Hobby), effectively severing access to business-critical tools.

**The decision was made to prioritize business continuity by decommissioning the local Docker containers to restore Odoo connectivity immediately.**

## 2. System Architecture & The Intersection

To understand the failure, we must analyze how the two systems intersected on the host machine.

### System A: The Corporate Requirement (Odoo)

- **Role:** Business ERP and Daily Operations.
- **Network Requirement:** The host machine must route traffic to a specific private IP range (e.g., 172.17.x.x or similar 172-block subnets) to reach the Odoo server.
- **Status: Protected / Mandatory.**

### System B: The Development Environment (WordPress)

- **Role:** Development Sandbox / Hobby Project.
- **Architecture:** Containerized via Docker (Engine + Compose).
- **Configuration:** \* Custom WP\_MEMORY\_LIMIT set to 512M (Resolved previous Elementor crash).
  - Running on local port 8080.
- **Status: Secondary / Adaptable.**

## 3. Root Cause Analysis (RCA)

The conflict was not caused by software incompatibility (Odoo vs. WordPress), but by **Network Routing Overlap**.

## The Mechanism of Failure

1. **Default Docker Behavior:** When the WordPress Docker container was initialized, Docker created a default bridge network (interface docker0). By default, Docker often assigns the subnet 172.17.0.0/16 to this bridge.
2. **The Intersection:** The Corporate Odoo server utilizes an IP address that falls within or routes through this same CIDR block (172.17.x.x).
3. **The Routing Decision:** When the user attempted to access Odoo, the Linux kernel checked its routing table. It saw a "local" connection (the Docker bridge) matching the requested IP range.
4. **The Result:** The system routed the request internally to the Docker container (which knew nothing about Odoo) instead of sending it out via the network card (LAN/WiFi) to the actual corporate server.

**Diagnosis:** The local development environment inadvertently hijacked the routing path required for the enterprise environment.

## 4. Resolution Protocol

To restore business operations, the following protocol was executed. This prioritizes the "Work" environment over the "Hobby" environment.

### Phase 1: Decommissioning the Conflict

*Objective: Remove the conflicting network interfaces without uninstalling the Docker engine entirely.*

#### 1. Stop Active Containers:

All running containers related to the WordPress stack (Database, PHPMyAdmin, WordPress Core) were halted to stop active processes.

```
docker stop $(docker ps -a -q)
```

#### 2. Remove Container Instances:

The containers were removed to free up the network bindings.

```
docker rm $(docker ps -a -q)
```

#### 3. Prune Docker Networks (Critical Step):

This was the decisive step. Stopping containers does not always remove the virtual network bridge. We forced the removal of the 172.17.0.0/16 bridge.

```
docker network prune
```

# Confirmed with 'y' to remove unused networks

#### 4. Service Restart:

The Docker daemon was restarted to clear cached routing tables.

```
sudo systemctl restart docker
```

## 5. Verification Results

Test Vector	Status	Notes
Odoo Connectivity	RESTORED	Access to http://172.17.x.x:8069 (or similar) is now successful. Traffic routes correctly via LAN.
WordPress Dev	OFFLINE	The development environment has been safely spun down. Data persists in volumes (if mapped), but service is inactive.
Host Stability	STABLE	Ubuntu 20.04 routing tables are normalized.

## 6. Future Recommendations (The "Safe" Path)

To re-introduce the WordPress development environment *without* breaking Odoo access again, one of the following architectures must be used:

### Option A: Configure Docker Subnets (Advanced)

Modify the Docker daemon configuration (`/etc/docker/daemon.json`) to force Docker to use a non-conflicting IP range (e.g., `192.168.50.0/24`).

- Pros: Keeps Docker fast and native.
- Cons: Requires modifying root configuration files.

### Option B: Virtual Machine Isolation (Recommended)

Run the WordPress Docker stack inside a Virtual Machine (e.g., VirtualBox) or a separate LXC container.

- Pros: Total network isolation. The VM gets one IP, and all Docker networking happens *inside* the VM, never touching the Host's routing table.
- Cons: Slightly higher system resource usage.

## 7. Implemented Long-Term Solution: Custom Docker Subnet Configuration

To verify "Option A" above and permanently fix the conflict between the Odoo 12 installation and Docker, the following configuration changes were applied to the host system. This ensures Docker avoids the `172.17.x.x` range entirely.

### Step 1: Stop Docker Service

The service was stopped to release all current network bindings.  
sudo systemctl stop docker

## Step 2: Edit Docker Daemon Configuration

We accessed the daemon configuration file:

```
sudo nano /etc/docker/daemon.json
```

## Step 3: Apply New IP Ranges

The following JSON configuration was added (or merged) into the file. This explicitly defines new IP ranges for Docker that do not conflict with the Odoo network.

```
{
  "bip": "172.26.0.1/16",
  "default-address-pools": [
    {
      "base": "172.80.0.0/16",
      "size": 24
    }
  ]
}
```

### Explanation of Settings:

- **bip:** Sets the default bridge network (docker0) to use 172.26.0.1/16 instead of the default 172.17.0.1.
- **default-address-pools:** Ensures any new user-defined networks (like those created by docker-compose) start from 172.80.0.0/16, preventing random assignment into the conflict zone.

### Result:

After restarting Docker (sudo systemctl start docker), the conflict was permanently resolved. Odoo remains accessible on its network, and Docker containers now run safely on 172.26.x.x.

**Signed:** System Administrator / Development Team