

The current version is #ident  
"@(#)\$Format:LocalFoodAI\_lanfr144:disaster\_recovery\_plan.md:Francois  
Lange:lanfr144@school.lu:2026/06/11 08:26:59:Francois  
Lange:lanfr144@school.lu:2026/06/11  
08:26:59:1701828b122e0c319e59134ca6511a42ecad9297::\$"

# Disaster Recovery & Backup Plan

This document outlines the backup and restore procedures, as well as the Disaster Recovery (DR) plan for the Local Food AI stack.

## 1. Backup Procedures

Given the massive 3GB+ size of the OpenFoodFacts dataset, backing up the entire MySQL data volume dynamically is resource-intensive. The strategy is split into **Code Backup** and **Data Backup**.

### 1.1 Source Code & App Configuration

The entire application infrastructure (Dockerfiles, Python scripts, configuration) is tracked in the Git repository. **Backup Command:** `git push origin main` Frequency: Triggered automatically by developers after every Sprint.

### 1.2 Database (MySQL) Backup

We use `mysqldump` to create a cold-standby physical backup of the user data and dietary profiles, while ignoring the massive, immutable OpenFoodFacts partition (which can be re-ingested from the source CSV).

**Backup Command:**

```
sudo docker exec food_project-mysql-1 mysqldump -u root -proot_pass foo
```

Frequency: Daily via a server-side cron job (`0 3 * * *`).

## 2. Restore Procedures

### 2.1 Database Restore (Warm Recovery)

If the database container crashes or the volumes are corrupted:

1. Stop the application container to prevent write conflicts: `sudo docker-compose stop app`
2. Wipe and re-initialize the MySQL container.
3. Restore the user tables from the SQL dump:

```
cat /backup/food_db_users_2026-05-12.sql | sudo docker exec -i food_pro
```

1. Restart the background ingestion script (`./data_sync.sh`) to rebuild the massive 3GB OpenFoodFacts `products_core` tables.

2. Restart the application: `sudo docker-compose start app`

## 3. Disaster Recovery (DR) Plan

### 3.1 Recovery Objectives

- **Recovery Time Objective (RTO):** 4 Hours (primarily bottlenecked by the 3-hour re-ingestion time of the CSV dataset if the core tables are lost).
- **Recovery Point Objective (RPO):** 24 Hours (User profiles and plates are backed up nightly).

### 3.2 High Availability & Failover Strategy

If deploying in the distributed Multi-Hypervisor PoC environment (Hyper-V / VirtualBox / WSL):

- **Ollama Node Failure:** The `app` is engineered to gracefully catch LLM connection timeouts. If the VirtualBox Ollama node dies, the Streamlit app will continue to function for standard Database lookups, returning a safe fallback message for AI evaluations.
- **Zabbix Node Failure:** The SNMP daemons run autonomously in each container. If the Zabbix telemetry server goes offline, the containers will safely drop the UDP traps without bottlenecking application performance.

### 3.3 Network/Server Disconnect & Local Fallback Mode

When the network to the remote host VM (`192.168.130.170`) or the Taiga server (`192.168.130.161`) is unavailable:

- **Resilient Single-Node Local Fallback:** The entire stack is completely containerized. By running `docker compose up -d` at the root of the workspace, MySQL, Ollama (with Llama 3.2:3B), SearXNG, Zabbix (Server, Web, Agent), Nginx, and the Streamlit App run entirely locally on the same host, avoiding external network dependencies.
- **Taiga Sync Bypass:** The `taiga_sync_final.py` execution is decoupled from core runtime. In disconnected mode, commits and user stories are managed through local Git branch state and manual tasks/walkthroughs, and sync scripts can be run once the connection is restored.
- **Dynamic Configuration Resolvers:** The application components, database migrations (`alembic/env.py`), and SNMP notification wrappers (`snmp_notifier.py`, `configure_zabbix_alerts.py`) automatically detect the local environment or fall back to container names (e.g. `mysql`, `zabbix-server`) rather than crashing on unreachable remote IPs.