

# Step 5: Professional Deployment with Docker

University: Tek-Up

Group: ING-4-SDIA

Tutor: Haythem Ghazouani

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Why Docker? . . . . .	2
<b>2</b>	<b>Phase 1: Dockerizing the Backend</b>	<b>2</b>
<b>3</b>	<b>Phase 2: Dockerizing the Frontend</b>	<b>3</b>
<b>4</b>	<b>Phase 3: Orchestration with Docker Compose</b>	<b>3</b>
<b>5</b>	<b>Launching the Stack</b>	<b>4</b>
5.1	Method 1: Command Line Interface . . . . .	4
5.2	Method 2: Docker Desktop (GUI) . . . . .	4
<b>6</b>	<b>Advanced Concepts: Ensembling</b>	<b>4</b>
<b>7</b>	<b>Useful Commands</b>	<b>4</b>
7.1	View Specific Logs . . . . .	4
7.2	Stop & Cleanup . . . . .	4
<b>8</b>	<b>Troubleshooting</b>	<b>4</b>

# 1 Introduction

## 1.1 Why Docker?

In this module, you built a Python backend and a React frontend. To run this project on another machine (e.g., your friend's laptop or a cloud server), you would normally have to:

1. Install Python 3.9
2. Install Node.js v18
3. Install dependencies ('pip install', 'npm install')
4. Solve OS-specific errors (Windows vs Linux paths)

**Docker** solves this by packaging the application and its entire environment (OS libraries, dependencies) into a lightweight unit called a **container**.

### Concept: Image vs. Container

- **Image:** The blueprint (read-only). Think of it as a Class in Python. It contains the code and libraries.
- **Container:** The running instance. Think of it as an Object. You can run multiple containers from one image.

## 2 Phase 1: Dockerizing the Backend

We create a file named `Dockerfile.backend` in the root directory.

```
1 # 1. Base Image: Start with a lightweight Linux + Python 3.9
2 FROM python:3.9-slim
3
4 # 2. Workdir: Create a folder inside the container
5 WORKDIR /app
6
7 # 3. Dependencies: Install gcc for some ML libraries
8 RUN apt-get update && apt-get install -y build-essential
9
10 # 4. Requirements: Copy only the file first (for caching)
11 COPY requirements.txt .
12 RUN pip install --no-cache-dir -r requirements.txt
13
14 # 5. Code: Copy your Python scripts and Data
15 COPY code/ ./code/
16 COPY data/ ./data/
17
18 # 6. Command: What runs when the container starts?
19 EXPOSE 8000
20 CMD ["python", "code/app.py"]
```

## Common Pitfall

Order matters! We copy ‘requirements.txt‘ **before** the code. This uses Docker’s “Layer Caching” mechanism. If you change your code but not your dependencies, Docker skips the slow ‘pip install’ step during the next build.

## 3 Phase 2: Dockerizing the Frontend

For React, we use a **Multi-Stage Build**. We don’t want to ship the heavy Node.js environment to production; we only want the compiled static files (HTML/CSS/JS).

```
1 # Stage 1: The Build Environment
2 FROM node:18-alpine AS build
3 WORKDIR /app
4 COPY frontend/package*.json ./
5 RUN npm install
6 COPY frontend/ ./ 
7 RUN npm run build
8 # This creates a 'dist/' folder with optimized files
9
10 # Stage 2: The Production Environment (Nginx)
11 FROM nginx:stable-alpine
12 COPY --from=build /app/dist /usr/share/nginx/html
13 EXPOSE 80
14 CMD ["nginx", "-g", "daemon off;"]
```

## 4 Phase 3: Orchestration with Docker Compose

Instead of running two manual ‘docker run‘ commands, we use **Docker Compose** to define the entire system.

File: docker-compose.yml

```
1 version: '3.8'
2
3 services:
4     # The Python Backend Service
5     backend:
6         build:
7             context: .
8             dockerfile: Dockerfile.backend
9         ports:
10            - "8000:8000" # Host Port : Container Port
11         volumes:
12            - ./data:/app/data # Persist data if container dies
13
14     # The React Frontend Service
15     frontend:
16         build:
17             context: .
18             dockerfile: Dockerfile.frontend
```

```
19 ports:
20   - "5173:80"    # Access dashboard at localhost:5173
21 depends_on:
22   - backend      # Make sure backend starts first
```

## 5 Launching the Stack

### 5.1 Method 1: Command Line Interface

To build and start all three services (Backend, Frontend, MLflow) in the background:

```
1 docker-compose up --build -d
```

### 5.2 Method 2: Docker Desktop (GUI)

For a visual overview of your project:

1. **Containers Tab:** Open Docker Desktop and click "Containers".
2. **Stack View:** Find the group `datasciencefopython2`.
3. **Monitoring:** You can see the status of each service, click the names to see logs, or use the "Open in Browser" shortcuts for the Frontend.

## 6 Advanced Concepts: Ensembling

We have implemented **Stacking**, where multiple base models (RF, XGBoost) contribute to a meta-model (Logistic Regression). This typically provides more stable and accurate results for complex datasets like Bank Churn.

## 7 Useful Commands

### 7.1 View Specific Logs

```
1 docker-compose logs -f backend
```

### 7.2 Stop & Cleanup

```
1 docker-compose down
```

## 8 Troubleshooting

**Issue:** "Connection Refused" between Frontend and Backend.

**Fix:** In Docker, 'localhost' refers to the container itself. To talk to another service, use its service name.

Example: In React, request `http://backend:8000` instead of `localhost:8000` (Note: This requires specific Nginx proxy setup for browser access, so for local dev, we stick to localhost mapping).