INSTALLATION & DEPLOYMENT GUIDE

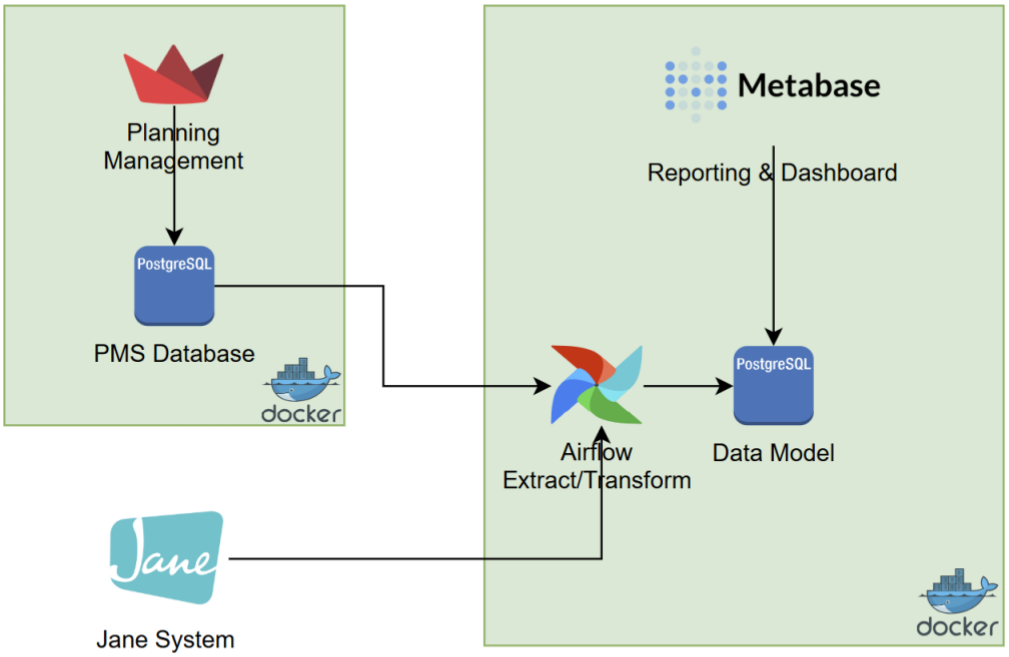
Software setup and application deployment
for Decision Support System

Andy N., Anthony P., Dawn D., Divyanshu F., Huan L.

Table of Contents

1 Introduction

This document provides a comprehensive guide to set up and deploy a **Decision Support System** (DSS) for **customer**, leveraging modern data tools and technologies. The primary goal of this system is to enhance clinic operations through streamlined data management, efficient ETL (Extract, Transform, Load) processes, and intuitive reporting dashboards.



Data Extraction - Apache Airflow
Database - PostgreSQL
Web App - Streamlit
Report - Metabase
Deployment - Docker

The system architecture, as illustrated, is designed to integrate seamlessly with existing tools, ensuring scalability, reliability, and ease of use. Key components of this architecture include:

1. **Planning Management System**:
    - Developed using **Streamlit** for user-friendly planning and management.
    - A robust **PostgreSQL** database serves as the foundation for storing and managing clinic staff planning and operational data.
    - Deployed via **Docker** for simplified setup and maintenance.
2. **ETL Pipeline**:
    - **Apache Airflow** automates the extraction and transformation of data from the clinic's **Jane System** and planning database.

- The pipeline ensures data consistency and accuracy, loading it into a data model tailored for performance monitoring and reporting.

3. **Performance Dashboards**:
    - A comprehensive dashboard built with **Metabase** provides insights into clinic performance, staff efficiency, and target achievement.
    - The data model, hosted on **PostgreSQL**, supports these reports with a well-structured schema optimized for analytical queries.

The deployment of this architecture relies on **Docker**, ensuring that each component operates in isolated and consistent environments, facilitating a smooth setup, updates, and scalability. By implementing this decision support system, Clinics will gain actionable insights into their operations, empowering data-driven decisions and enhancing overall performance.

2 Prerequisites

- **Operating System**: Windows 10 or higher.
- **RAM**: Minimum 8 GB (16 GB recommended).
- **CPU**: Minimum 4 cores.
- **Software**:
    - **Docker Desktop**: Latest version.
    - **Docker Images:**
        - postgres:latest
        - python:3.9-slim
        - extending_airflow:latest
        - metabase:latest

3 Docker Installation

Download Docker Desktop:

- Visit Docker's official website and download the installer for Windows.

Install Docker Desktop:

- Run the downloaded installer.
- Follow the prompts to complete the installation.

Verify Installation:

- Open a terminal and run:

```
docker --version
```

```
Docker version 24.0.6, build ed223bc
```

- Ensure Docker is running before proceeding.

4 Planning Management System (PMS)

4.1 PMS Database

PostgreSQL Deployment on Docker

1. Run the command to configure PostgreSQL for the **Planning Management System**:

```
docker pull postgres
docker run --name pms_postgres -e POSTGRES_PASSWORD=your_password -p 5434:5432 -d postgres
```

Verify that the database container is running:

```
docker ps -f "name=pms_*"
```

```
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS          PORTS                    NAMES
ae27a8199d7f   postgres   "docker-entrypoint.s…"   4 seconds ago    Up 2 seconds    0.0.0.0:5434->5432/tcp   pms_postgres
```

2. Create table structure on pms database:

Connect to database:

```
docker exec -it pms_postgres psql -U postgres
```

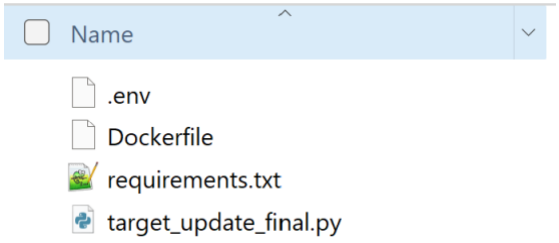
Execute scripts to create tables:

```
-- Create the database
CREATE DATABASE dashboard;

-- Connect to the database
\c dashboard;

-- Create the schema
CREATE SCHEMA IF NOT EXISTS planning1;

-- Create the practitioner table
CREATE TABLE IF NOT EXISTS planning1.practitioner (
    practitioner_id INTEGER NOT NULL,
    practitioner_name CHARACTER VARYING(255),
    employee_type CHARACTER VARYING(255),
    clinic_location CHARACTER VARYING,
    bill_rate_standard INTEGER,
```

```
    bill_rate_special INTEGER,
    manager_name CHARACTER VARYING(50),
    CONSTRAINT practitioner_pkey PRIMARY KEY (practitioner_id)
);

-- Create the statutory_holidays table
CREATE TABLE IF NOT EXISTS planning1.statutory_holidays (
    holiday_date DATE NOT NULL,
    holiday_name CHARACTER VARYING(100),
    CONSTRAINT statutory_holidays_pkey PRIMARY KEY (holiday_date)
);

-- Create the target_update table
CREATE TABLE IF NOT EXISTS planning1.target_update (
    practitioner_id INTEGER,
    practitioner_name CHARACTER VARYING(255),
    target_date TIMESTAMP,
    target_hour DOUBLE PRECISION,
    updated_at TIMESTAMP,
    CONSTRAINT unique_practitioner_date UNIQUE (practitioner_id, target_date)
);
```

4.2 PMS Application

List of files for application deployment:



- **.env**: store database connection credentials
- **Dockerfile**: to build docker image
- **Requirements.txt**: to list out python libraries
- **Target_update_final.py**: application code

This is the content of Dockerfile and requirement.txt

Dockerfile

```
# Use the official Python image
FROM python:3.9-slim

# Set the working directory
WORKDIR /app

# Copy the application code
COPY . /app

# Install system dependencies
```

```
RUN apt-get update && apt-get install -y libpq-dev gcc

# Install Python dependencies
RUN pip install --no-cache-dir -r requirements.txt

# Expose Streamlit default port
EXPOSE 8501

# Set Streamlit environment variables
ENV STREAMLIT_SERVER_PORT=8501
ENV STREAMLIT_SERVER_ADDRESS=0.0.0.0

# Command to run the app
CMD ["streamlit", "run", "target_update_final.py"]
```

requirement.txt

```
streamlit
pandas
psycopg2-binary
matplotlib
streamlit-option-menu
python-dotenv
```

Build docker image: run cmd at source code folder

```
docker build -t pms-app .
```

```
PS D:\OneDrive\DANA4850-Capstone\target_management> docker build -t pms-app .
[+] Building 55.5s (10/10) FINISHED                                                                    docker:default
 => [internal] load .dockerignore                                                                                0.0s
 => => transferring context: 2B                                                                                  0.0s
 => [internal] load build definition from Dockerfile                                                             0.1s
 => => transferring dockerfile: 596B                                                                             0.0s
 => [internal] load metadata for docker.io/library/python:3.9-slim                                               1.6s
 => [1/5] FROM docker.io/library/python:3.9-slim@sha256:4ee0613170ac55ebc693a03b6655a5c6f387126f6bc3390e739c2e6c3  1.7s
 => => resolve docker.io/library/python:3.9-slim@sha256:4ee0613170ac55ebc693a03b6655a5c6f387126f6bc3390e739c2e6c3  0.0s
 => => sha256:2f9d412a81d0e26acee1ddd6db40e1465d8d3b0070113d18ba73086eb54778c3 1.75kB / 1.75kB                    0.0s
 => => sha256:473b3636d11e17647bf9e36816053317dd5a6667e8c2d0b8e1110c1a89249133 5.28kB / 5.28kB                    0.0s
 => => sha256:c3edffebd7235334426d893f9c7b8aa09ea786e71b6bcdef6860a593a7f8f21b 3.32MB / 3.32MB                    0.6s
 => => sha256:c07f3a6e2bd8775018ae44783bcef58a2ca4c4557fdfeb339c00ba9cf5388dc4 14.93MB / 14.93MB                  0.8s
 => => sha256:e898f07f768a6be80b5271f2ab022b43a7110379fb29a6c01e9d272754adb300 250B / 250B                        0.3s
 => => sha256:4ee0613170ac55ebc693a03b6655a5c6f387126f6bc3390e739c2e6c337880ef 10.41kB / 10.41kB                  0.0s
 => => extracting sha256:c3edffebd7235334426d893f9c7b8aa09ea786e71b6bcdef6860a593a7f8f21b                         0.2s
 => => extracting sha256:c07f3a6e2bd8775018ae44783bcef58a2ca4c4557fdfeb339c00ba9cf5388dc4                         0.7s
 => => extracting sha256:e898f07f768a6be80b5271f2ab022b43a7110379fb29a6c01e9d272754adb300                         0.0s
 => [internal] load build context                                                                                0.1s
 => => transferring context: 48.99kB                                                                             0.0s
 => [2/5] WORKDIR /app                                                                                           0.1s
 => [3/5] COPY . /app                                                                                            0.0s
 => [4/5] RUN apt-get update && apt-get install -y libpq-dev gcc                                                22.0s
 => [5/5] RUN pip install --no-cache-dir -r requirements.txt                                                    28.0s
 => exporting to image                                                                                           1.9s
 => => exporting layers                                                                                          1.9s
 => => writing image sha256:cc69424786ee8402c7e46c23aa23e9babc58afc76880705050f77acce2f28979                      0.0s
 => => naming to docker.io/library/pms-app                                                                       0.0s
```

Run container to deploy app:

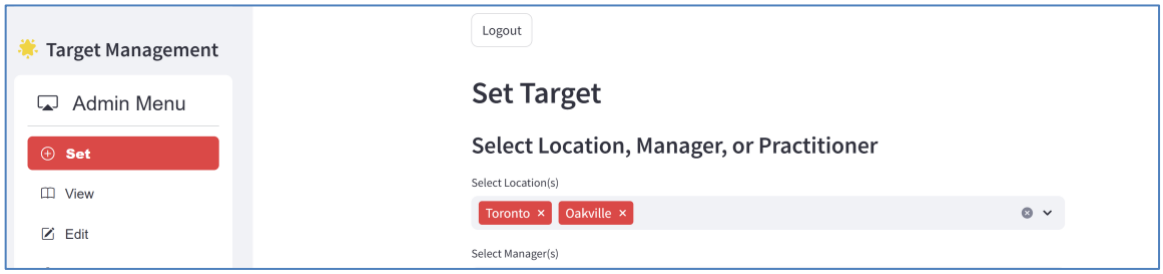
```
docker run -d -p 8501:8501 --name pms-app pms-app
```

Verify if the container is running:

```
docker ps -f "name=pms-app*"
```

```
PS D:\OneDrive\DANA4850-Capstone\target_management> docker ps -f "name=pms-app*"
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS          PORTS                    NAMES
c943663d7afe   pms-app    "streamlit run targe…"   46 seconds ago   Up 45 seconds   0.0.0.0:8501->8501/tcp   pms-app
```

Test planning management app on web browser:



5 Data Model

To save resources, the data model is also deployed on pms database with a different schema.

Connect to database:

```
docker exec -it pms_postgres psql -U postgres
```

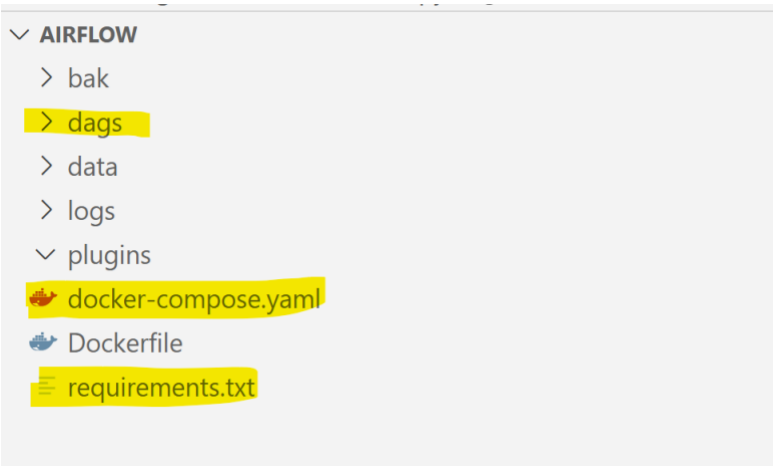
Execute scripts to create tables:

```
-- Connect to the database
\c dashboard;

-- Create the schema
CREATE SCHEMA IF NOT EXISTS model;

-- Create the Dim_Date table in PostgreSQL
CREATE TABLE model.Dim_Date (
    date DATE PRIMARY KEY,
    day_of_week VARCHAR(10),
    week_of_year INT,
    month INT,
    quarter INT,
    year INT
);

-- Populate the Dim_Date table with dates from 2020-01-01 to 2030-12-31
INSERT INTO model.Dim_Date (date, day_of_week, week_of_year, month, quarter, year)
SELECT
    d::DATE AS date,
    TO_CHAR(d, 'Day') AS day_of_week,
    EXTRACT(WEEK FROM d)::INT AS week_of_year,
    EXTRACT(MONTH FROM d)::INT AS month,
    EXTRACT(QUARTER FROM d)::INT AS quarter,
```

```
    EXTRACT(YEAR FROM d)::INT AS year
FROM generate_series('2020-01-01'::DATE, '2030-12-31'::DATE, INTERVAL '1 day') AS d;



CREATE TABLE model.Dim_Practitioner (
    practitioner_dim_id SERIAL PRIMARY KEY,
    practitioner_id INT NOT NULL,
    practitioner_name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    contract_type VARCHAR(50),
    manager_name VARCHAR(100),
          location_id INT,
          location_name VARCHAR(100),
    effective_start_date DATE,
    effective_end_date DATE,
    is_current BOOLEAN DEFAULT TRUE
);



CREATE TABLE model.Dim_Item (
    item_id SERIAL PRIMARY KEY,
    item_name VARCHAR(100) NOT NULL,
    department VARCHAR(100),
    category VARCHAR(100)
);


-- Create the Dim_Location table in PostgreSQL
CREATE TABLE model.Dim_Location (
    location_id SERIAL PRIMARY KEY,
    location_name VARCHAR(100) NOT NULL,
    address VARCHAR(255)
);


CREATE TABLE model.Fact_Performance (
    date DATE NOT NULL,
    practitioner_dim_id INT NOT NULL,
    location_id INT NOT NULL,
    target_hour DECIMAL(5, 1),
    actual_hour DECIMAL(5, 1),
    total_billing DECIMAL(10, 1)
);


CREATE TABLE model.Fact_Appointments (
    date DATE NOT NULL,
    practitioner_dim_id INT NOT NULL,
    location_id INT NOT NULL,
          item_id INT,
          number_appoiments INT,
    actual_hour DECIMAL(5, 1),
    total_billing DECIMAL(10, 1)
);
```

6 Airflow for Data Pipelines (ETL)

We have the airflow source code project as follows:



Main components are:

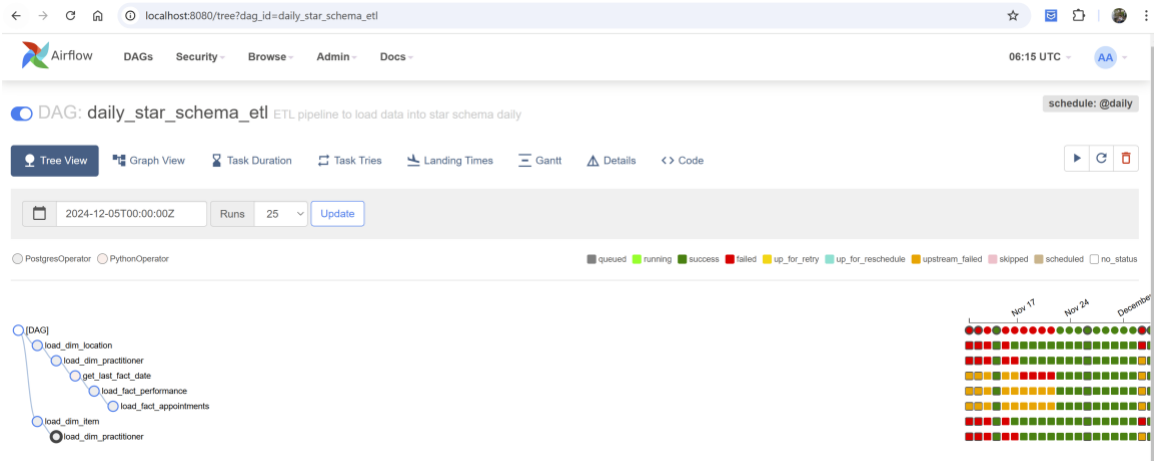
- **Dags** folder: store all the data pipelines source code
- **Docker-compose.yaml**: declare all the services to run airflow
- **Requirements.txt**: declare all python libraries used

Build airflow on docker: run cmd in the source code project

```
docker-compose up airflow-init
docker-compose up -d
```

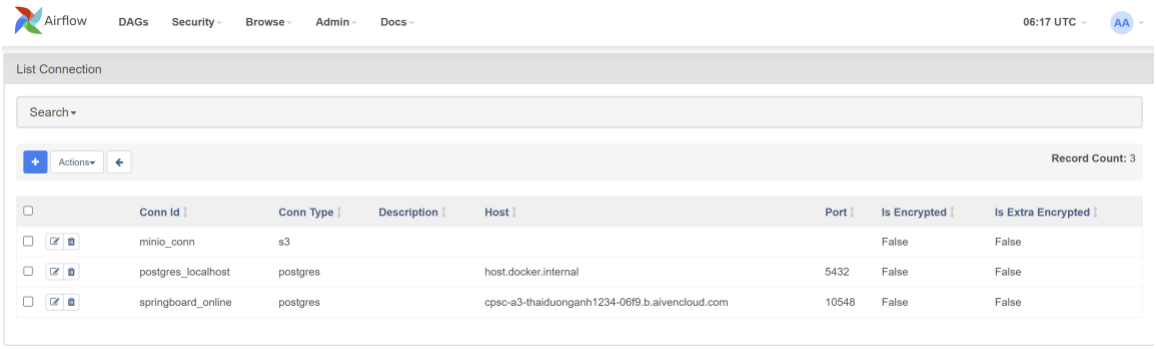
Verify if all the airflow containers are running:

Verify on web browser: go to http://localhost:8080



Setup database connection for Airflow:

Go to **Admin → Connections**:



Add new connections or edit existing connections and save.

7 Metabase for Dashboards

7.1 Metabase Installation on Docker

1. Use `docker-compose.yaml` to configure Metabase:

```yaml
version: "3.8"

services:
  metabase:
    image: metabase/metabase:latest
    container_name: metabase
    ports:
      - "3000:3000"
    volumes:
      - metabase_data:/metabase-data

volumes:
  metabase_data:
```

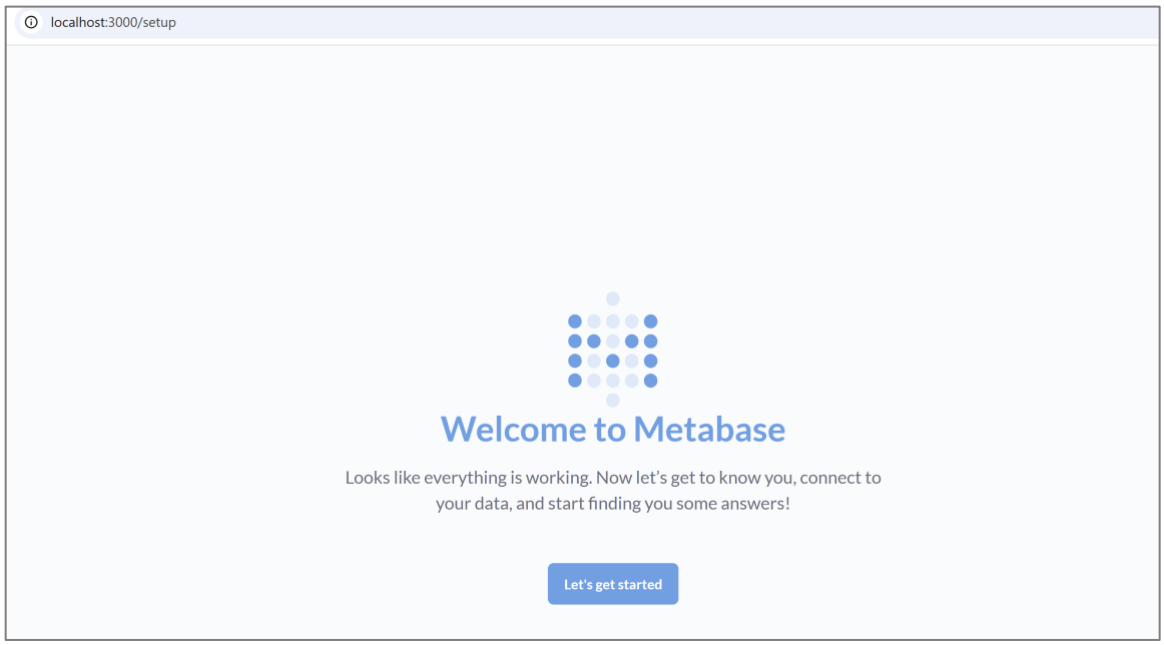
2. Start the Metabase service:

```
docker-compose up -d metabase
```

Verify if the docker container is running:

```
PS D:\OneDrive\DANA4850-Capstone\metabase> docker-compose ps
NAME       IMAGE                     COMMAND                  SERVICE    CREATED              STATUS              PORTS
metabase   metabase/metabase:latest  "/app/run_metabase.sh"   metabase   About a minute ago   Up About a minute   0.0.0.0:3000->3000/tcp
```

Verify on web browser: http://localhost:3000/



7.2 Import Metabase Report

1. Replace the metabase.mv.db file in the folder /metabase.db in the container with the file provided from the development environment.
2. Restart Metabase:

```
docker restart metabase
```

3. Access Metabase at http://localhost:3000 and verify the imported reports.