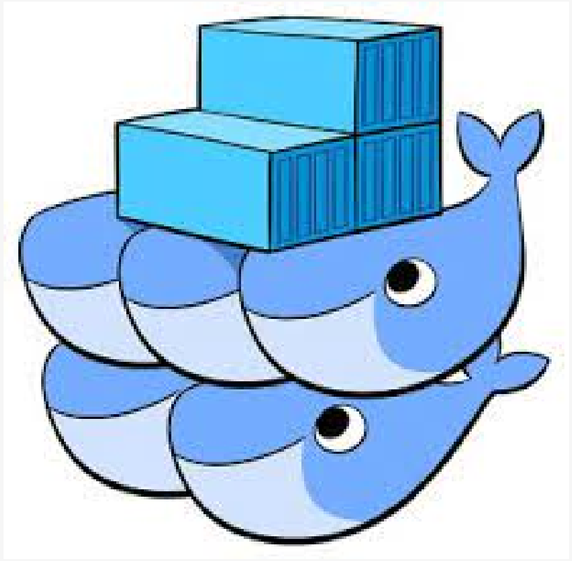

Docker SWARM

Ex 01

3000

Submit your codes and the URL of the smart website you have designed.

Generated by ChatGPT 4

Developing and Deploying a Smart Website Using Flask, Docker, Docker Swarm, and a Database

Objective:

The goal of this exercise is for students to collaboratively develop a smart website using the Flask framework, implement a machine learning model, integrate a user registration and login system with a database, containerize the application with Docker, and deploy it using Docker Swarm on a remote host. Each team will work together to build, test, and deploy a fully functional web application that interacts with users, stores data in a database, and is accessible via a cloud-based host.

Team Structure:

Each team will consist of 5 members. The tasks will be divided to ensure all team members contribute to the development, database integration, containerization, and deployment of the project. The smart website will collect user inputs, process them using a machine learning model, and return results while supporting user registration, login, and personalized data storage.

Part 1: Frontend Development (Team Member 1)

Role: Frontend Developer

Task :

- Design the user interface for the web application using HTML, CSS, and Bootstrap.
- Create the following pages:
    - **Home page:** A landing page that introduces the app and its features.
    - **Registration page:** A form for users to register with fields like username, email, and password.
    - **Login page:** A form for user authentication.
    - **Input page:** A page for users to input data for the model's prediction.
    - **Result page:** Displays the output from the machine learning model.
    - **Profile page:** Shows personalized information (e.g., previous predictions) for logged-in users.
    - **Error pages:** Handle common errors like incorrect input or unauthorized access.

Part 2: Backend Development and Authentication (Team Member 2)

Role: Backend Developer

Task :

- Set up Flask routes and views for the frontend pages.
- Implement a user registration system using SQLAlchemy to store user credentials in a SQLite (or any chosen) database.
- Implement a login system with user authentication. Use Flask sessions to manage logged-in users.
- Secure the input page and result page so that only authenticated users can access them.
- Handle backend logic that processes user input from the input page and sends the data to the machine learning model for predictions.
- Implement error handling for login failures, invalid input, or unauthorized access attempts.

Part 3: Database Integration (Team Member 3)

Role: Database Engineer

Task :

- Design and set up a SQLite (or other) database to store user data such as:
  - User credentials (username, email, password).
  - Input history and results for personalized user experience (optional).
- Ensure SQLAlchemy is used to connect the Flask app to the database.
- Create tables for:
  - User: To store registration details.
  - Prediction history: Optional, to store users' past predictions.
- Write backend logic to query the database for user authentication and save model predictions (optional).
- Ensure that proper data validation and security measures are in place for handling sensitive data like passwords (e.g., hash passwords before storing them).

Part 4: Machine Learning Model (Team Member 4)

Role: Machine Learning Engineer

Task :

- Select and train a machine learning model based on the use case (e.g., predicting house prices, diagnosing diseases).
- Preprocess the data and train the model using Python libraries like scikit-learn, TensorFlow, or PyTorch.
- Write a model.py script that loads the saved model and processes inputs from the backend to return predictions.
- Ensure the model is efficient, and return accurate results based on user input.

Part 5: Containerization and Deployment (Team Members 4 & 5)

Role: MLOps Engineer / DevOps Engineer

Task :

- Write a Dockerfile to containerize the Flask app, including dependencies like Flask, SQLAlchemy, and any machine learning libraries.
- Ensure that the app works in a local Docker container and interacts with the database (e.g., SQLite) and the machine learning model.
- Write a docker-compose.yml file to define multiple services, such as:
    - The Flask app.
    - The database (if needed).
- Test the containerized app locally to ensure all components work together.
- Set up a cloud host (e.g., DigitalOcean, AWS EC2) and install Docker and Docker Swarm.
- Write a docker-stack.yml file to deploy the application on Docker Swarm, ensuring that it is scalable.
- Deploy the app on the host using Docker Swarm, ensuring the database and app work in the hosted environment.
- Ensure the app is publicly accessible via the host's IP or domain.
- Test the final deployment to verify all features (registration, login, prediction, etc.) work as expected.

Project Workflow

1. Planning:

- Teams plan the project structure, define the database schema, design API endpoints, and decide on model requirements.
- Break down the tasks for frontend, backend, database, and deployment to ensure smooth collaboration.

2. Development:

- Team members work on their individual tasks while collaborating on integration.
- Regular testing is done to ensure that all parts work together (e.g., forms correctly send data to the backend, models return predictions).

3. Containerization:

- Once the Flask app works locally, the Docker specialist ensures it runs inside Docker containers.
- Testing with docker-compose locally to make sure multiple services (Flask app, database) run together.

4. Deployment:

- The DevOps engineer deploys the app to a cloud host using Docker Swarm.
- The deployed app is tested for functionality, including user registration, login, predictions, and database interactions.

5. Final Submission:

- Each team submits the publicly accessible URL of the deployed app, along with project documentation that includes:

1. Instructions for accessing and testing the live app.
2. Details on setting up the app locally, including Docker configurations.
3. Database schema and model details.

This exercise gives students hands-on experience in full-stack web development, user authentication, database integration, Docker containerization, and cloud deployment using Docker Swarm.

Best Wishes :)