Creating a Docker Image, Uploading to DockerHub, and Server Changes

This documentation provides a comprehensive guide to creating a Docker image for the Open WebUI application, uploading it to DockerHub, and detailing the critical changes made on the server to resolve database schema issues and ensure seamless application functionality. Key updates include inspecting and modifying the SQLite database, resolving a missing column error, rebuilding and restarting Docker containers, and validating the integration of litellm and other services. This process ensures the application is properly configured, error-free, and ready for deployment.

Server Implemented On:

1) (ec2-13-236-50-121.ap-southeast-2.compute.amazonaws.com)
2) (ec2-13-237-148-39.ap-southeast-2.compute.amazonaws.com)

(Site: https://chat.techpeek.in/, http://13.236.50.121:8080/)

1) Creating the Docker Image:

❖ To create a Docker image for the Open WebUI application:

- Navigate to the Project Directory: `cd ~/open-webui-latest-version`
- Build the Docker Image: Run the following command to build the Docker image: `sudo docker build -t saif233/open-webui:latest .`
- Verify the Image: List all Docker images to ensure it was built successfully: `docker images`

2) Uploading to DockerHub:

❖ To upload the Docker image to DockerHub:

- Login to DockerHub: `docker login` (Enter your DockerHub credentials when prompted.)
- Push the Docker Image: Push the newly created Docker image to DockerHub: `sudo docker push saif233/open-webui:latest`
- Verify the Image on DockerHub: Log in to your DockerHub account and confirm that the image is available in your repository.

3) Additional Steps in the Deployment Process:

- Up until the steps involving Docker image creation and pushing to DockerHub, all commands were executed on

ec2-13-236-50-121.ap-southeast-2.compute.amazonaws.com, the server hosting the Open WebUI latest version.

4) Pulling the Image on the Target Server:

- Switch to the target server, ec2-13-237-148-39.ap-southeast-2.compute.amazonaws.com, which hosts the Open WebUI main version.
- Navigate to the project directory: cd ~/open-webui-main
- Pull the Docker image from DockerHub: sudo docker pull saif233/open-webui:latest
- Spin up the containers: sudo docker compose up -d

5) Docker Compose YAML Configuration:

- On the target server (Open WebUI Main), the docker-compose.yaml file specifies the services and the image to be used. Below is the configuration:

➔ image: saif233/open-webui:latest

6) Configuration Updates:

- The config.yaml file on the target server was updated to reflect the same configurations used in Open WebUI latest version.

7) Changes Made on the Server:

- Here are the changes made on the server during this session:

1. Database Schema Update:

- Inspected the SQLite database (webui.db) located at backend/data/.
- Verified that the user table was missing the verified column.
- Added the verified column to the user table using:
- ALTER TABLE user ADD COLUMN verified BOOLEAN DEFAULT 0;

2. Configuration and Database Inspection:

- Identified the database file (webui.db) as the source of the issue.
- Used the sqlite3 command-line tool to inspect and modify the database schema.

3. Docker and YAML Updates:

- **Investigated docker-compose.yaml and related Docker configurations to verify and rebuild containers.**
- **Restarted the Docker containers to reflect database schema updates:**
  docker-compose down
  docker-compose up --build

4. Litellm Container:

- **Located existing containers such as litellm and ensured proper integration with Open WebUI.**
- **Documentation for commands: (https://docs.openwebui.com/tutorials/migration/)**

5. Error Resolution:

- **Resolved the error sqlite3.OperationalError: no such column: user.verified by adding the missing column.**
- **Tested the /api/config endpoint to confirm the fix.**

6. Constants.ts:

- **Added three API calls for litellm, caseprediction and legalsearch**
- **export const LITELLM_API_BASE_URL = `${WEBUI_BASE_URL}/litellm/api`; //TO CALL OUR TECHPEEK M1, M2 & M3 MODELS**
- **export const LEGAL_API_BASE_URL = `${WEBUI_BASE_URL}/api`; // THE ACTUAL API CALL TO LEGALSEARCH**
- **export const CASE_API_BASE_URL = `${WEBUI_BASE_URL}/api`; //THE ACTUAL API CALL TO CASEPREDICTION**

7. TechpeekMessageInput.svelte:

- **Similar as MessageInput.svelte I created a file TechpeekMessageInput.svelte on the same path src/lib/components/chat/TechpeekMessageInput.svelte to support our features as well the props passed to work with caseprediction and legalsearch pointing to TechpeekMessageInput bar function.**

8. Static Dir:

- **All png inside the static dir was replaced with our Techpeek AI logo with same name configurations and not to disturb any internal configurations for processing the same name png and successfully bringing up our Techpeek logo.**

9. Development:

- **For development purposes we always need to run “sudo npm run dev” so that each changes gets reflected immediately unlike docker where we rebuild the image for each changes and it’s also essential to run backend using “bash start.sh” because frontend only starts if backend is also running but what if there’s a scenario where 8080 port is already occupied in that case on this path backend/start8081.sh script was made where your backend will be allocated to 8081 if 8080 port is already occupied “*`PORT="${PORT:-8081}"`*.**

10. casepredictionmain.py:

- **Initially the path is hardcoded as *`# MODEL_PATH = "/home/ubuntu/open-webui/backend/open_webui/apps/caseprediction/"`* but it is later replaced with a dynamically generated path using os.path.join and os.path.dirname(__file__) This change ensures that the path to the model is relative to the location of the current script, making the application more portable and adaptable to different environments or deployment setups, as it no longer relies on a fixed directory structure. And also as the fast api is initialized by router = APIRouter() we have exported & mounted caseprediction in the primary backend/open_webui/main.py file as - from open_webui.apps.caseprediction.main import router as caseprediction_router, app.include_router(caseprediction_router, prefix="/api/caseprediction", tags=["Case Prediction"]).**

11. Open WebUI to Techpeek AI:

- **We replaced all mentions of "Open WebUI" with "Techpeek AI" in the project. This was done using the search-and-replace functionality in VS Code.**

12. Webui.db:

- **To ensure the backend starts fresh from scratch, delete the `webui.db` file located at `backend/data/webui.db`. Please note that this action will remove all users, deployed models, and reset all connections within the application. Before proceeding, make sure to securely save and back up your configurations to avoid losing important data.**

13. config.py:

- **We commented out references to `favicon.png`, `splash.png`, and other related assets in `config.py` to ensure that when developers run `bash start.sh`, the Techpeek AI logo is not replaced with the Open WebUI logo and renders correctly.**

14. Special Token Removal:

- **We also removed the special token references from svelte files to ensure that any branding or unique identifiers tied to Open WebUI were fully replaced, aligning the application with Techpeek AI's branding.**

15. Open WebUI Documentation:

- **Understanding Open WebUI from the ground up is crucial for gaining familiarity with its core technologies, including Svelte for the frontend and FastAPI for the backend. This foundational knowledge not only equips you with the skills needed to navigate and modify the application effectively but also helps you comprehend its internal server configurations, directory structures, and key features. By delving into these aspects, you will develop a comprehensive understanding of the system, enabling you to troubleshoot issues, implement new functionalities, and contribute to the server's development with confidence and efficiency. This foundational expertise lays the groundwork for making informed decisions and delivering impactful contributions to the project. (https://docs.openwebui.com/)**

- **Ibrahim Sultan**