Docker Compose Watch for Development in Danswer

This documentation details the steps required to configure a development environment with Docker Compose watch mode in the Danswer application. It includes setting up docker-compose.dev.yml for development, creating a development-specific Dockerfile, and with given options like rebuild, sync and sync+restart. Each step ensures efficient live syncing of code changes and an optimal development workflow.

Server Implemented on:
(http://ec2-54-79-231-211.ap-southeast-2.compute.amazonaws.com)

(Site: http://54.79.231.211/)
(Dev_Site: http://localhost:3000)

1) ***docker-compose.dev.yml:***

Path: cd danswer/deployment/docker_compose/docker-compose.dev.yml

Summary: This **file defines** the **services** and **configuration specifically** for the **development environment**. The **setup** allows **live syncing** of **changes** without the **need** to **rebuild** the **entire image** each time. The **watch feature** was **implemented** for the **web_server service** to ensure that all **frontend changes** are **instantly reflected** on the **live site**. I **experimented** with **different watch actions**, including **sync**, **sync+restart**, and **rebuild**, to **find** the **most optimal setup**. After **testing**, I **determined** that the **sync method** is the **best choice** for **frontend development**, as it **efficiently updates changes** without **interrupting** the **service**.

Code with action set to sync:

```
 web_server:
    image: danswer/danswer-web-server:${IMAGE_TAG:-latest}
    build:
      context: ../../web
      dockerfile: Dockerfile.dev - - > (Currently Targeting to Dockerfile.dev within the
web dir not to Dockerfile)
      args:
        -
NEXT_PUBLIC_DISABLE_STREAMING=${NEXT_PUBLIC_DISABLE_STREAMING:-
false}
        -
NEXT_PUBLIC_NEW_CHAT_DIRECTS_TO_SAME_PERSONA=${NEXT_PUBLIC_N
EW_CHAT_DIRECTS_TO_SAME_PERSONA:-false}
```

```
      -
NEXT_PUBLIC_POSITIVE_PREDEFINED_FEEDBACK_OPTIONS=${NEXT_PUBLIC
_POSITIVE_PREDEFINED_FEEDBACK_OPTIONS:-}
      -
NEXT_PUBLIC_NEGATIVE_PREDEFINED_FEEDBACK_OPTIONS=${NEXT_PUBLIC
_NEGATIVE_PREDEFINED_FEEDBACK_OPTIONS:-}
      - NEXT_PUBLIC_DISABLE_LOGOUT=${NEXT_PUBLIC_DISABLE_LOGOUT:-}
      -
NEXT_PUBLIC_DEFAULT_SIDEBAR_OPEN=${NEXT_PUBLIC_DEFAULT_SIDEBAR
_OPEN:-}

      # Enterprise Edition only
      - NEXT_PUBLIC_THEME=${NEXT_PUBLIC_THEME:-}
      # DO NOT TURN ON unless you have EXPLICIT PERMISSION from Danswer.
      -
NEXT_PUBLIC_DO_NOT_USE_TOGGLE_OFF_DANSWER_POWERED=${NEXT_PU
BLIC_DO_NOT_USE_TOGGLE_OFF_DANSWER_POWERED:-false}
   depends_on:
     - api_server
   restart: always
   environment:
     - INTERNAL_URL=http://api_server:8080
     - WEB_DOMAIN=${WEB_DOMAIN:-}
     - THEME_IS_DARK=${THEME_IS_DARK:-}
     - DISABLE_LLM_DOC_RELEVANCE=${DISABLE_LLM_DOC_RELEVANCE:-}

     # Enterprise Edition only
     -
ENABLE_PAID_ENTERPRISE_EDITION_FEATURES=${ENABLE_PAID_ENTERPRIS
E_EDITION_FEATURES:-true}
     -
NEXT_PUBLIC_CUSTOM_REFRESH_URL=${NEXT_PUBLIC_CUSTOM_REFRESH_
URL:-}
   develop:
     watch:
       - action: sync
         path: ../../web
         target: /app
         ignore:
           - node_modules/
       - action: rebuild
         path: ../../web/package.json
```

Code with action set to rebuild:

```
  develop:
    watch:
      - action: rebuild
        path: ../../web
        ignore:
          - node_modules/
```

Code with action set to sync+restart:

```
  develop:
    watch:
      - action: sync+restart
        path: ../../web
        target: /app
        ignore:
          - node_modules/
      - action: rebuild
        path: ../../web/package.json
```

- **Rebuild: While rebuild** updates **the web_server container**, it requires **Docker** to **recreate** the **container** each **time** whenever a **modification** is **made**, which can **take** a **bit** of **time**. This **delay makes** it less **ideal** for **fast-paced development workflows**, as it **disrupts** the **flow** by causing a **noticeable refresh time**.

- **Sync+Restart:** This **method**, which **syncs changes** and **restarts** the **container**, is only **necessary** when **making changes** to both **frontend** and **backend** files that are **interdependent**. For **example**, if the **backend** is **set up** with a **multi-stage build** process where files **from** the **frontend** (such as **assets**) are **copied** into the **backend** at the **build stage** (using a **builder stage**), then **sync+restart** can **ensure** that **backend dependencies** on **frontend code** are **accurately reflected**. **However**, for **frontend-only updates**, **sync+restart** is an **unnecessary step** that **introduces avoidable delay**.

- In **summary**, **sync** alone is the **optimal method** for **frontend development** in **Danswer**, **providing real-time updates** without the **added delay** of **container restarts** or **rebuilds**.

2) Dockerfile.dev:

Path: cd danswer/web/Dockerfile.dev

This Dockerfile is **tailored specifically** for **development**. It **omits production-related optimizations from the original Dockerfile** to **enable quicker builds** and **live updates**.

Code:

```
# Development Dockerfile (Dockerfile.dev)

# Use the base node image for development
FROM node:20-alpine AS base

LABEL com.danswer.maintainer="founders@danswer.ai"
LABEL com.danswer.description="This image is the web/frontend container of
Danswer which \
    contains code for both the Community and Enterprise editions of Danswer. If
you do not \
    have a contract or agreement with DanswerAI, you are not permitted to use the
Enterprise \
    Edition features outside of personal development or testing purposes. Please
reach out to \
    founders@danswer.ai for more information. Please visit
https://github.com/danswer-ai/danswer"

# Set up the working directory
WORKDIR /app

# Copy only package.json and package-lock.json initially to leverage Docker
cache
COPY package.json package-lock.json ./

# Install dependencies
RUN npm install

# Copy the entire source code
COPY . .

# Disable Next.js telemetry for development
ENV NEXT_TELEMETRY_DISABLED=1

# Set environment variables for development mode, adjusting as needed
```

```
ARG NEXT_PUBLIC_DISABLE_STREAMING
ENV
NEXT_PUBLIC_DISABLE_STREAMING=${NEXT_PUBLIC_DISABLE_STREAMING}

ARG NEXT_PUBLIC_NEW_CHAT_DIRECTS_TO_SAME_PERSONA
ENV
NEXT_PUBLIC_NEW_CHAT_DIRECTS_TO_SAME_PERSONA=${NEXT_PUBLIC_N
EW_CHAT_DIRECTS_TO_SAME_PERSONA}

# Run the development server
CMD ["npm", "run", "dev"]
```

3) *Start and Build Services:*

To start your **services** with the **new development configuration**, use the **following command**. This **command builds** and **recreates** the **containers** from the **docker-compose**.**dev**.**yml configuration**.

Code:

```
sudo docker compose -f docker-compose.dev.yml -p danswer-stack up -d --build
--force-recreate
```

4) *Enable Docker Watch Mode for Live Syncing:*

Docker's watch feature enables live syncing of **changes**, making it **easier** to see **updates** on the **site** without **needing** to **rebuild the image**.

Code:

```
sudo docker compose -f docker-compose.dev.yml -p danswer-stack watch
```

5) *Stop and Remove All Docker Containers:*

To clean up all **running containers** and **remove** any **stopped containers**, use **these commands**. This is **helpful** when you **need** to **free up memory** or **reset** the **environment**.

Code:

- `sudo docker stop $(sudo docker ps -aq)`
- `sudo docker rm $(sudo docker ps -aq)`

6) Troubleshooting: Memory Shortage:

- **During** the **setup process**, I **encountered memory shortages** due to **high memory consumption** by **Hot Reloading** via **Watch feature**. Here are a **few tips** to **manage memory usage effectively**:

- **Limit Memory Usage** in **Docker Compose**: You **can** set **memory limits** for **each container** in **docker-compose**.**dev**.**yml** to **prevent Docker** from **consuming** too **much memory**.

- **Another useful tip** when **configuring** the **sync method** in **docker-compose**.**dev**.**yml** is to **specify** only the **files** or **directories** you **need** to **sync**, **rather** than the **entire project directory**. This **can significantly** improve **performance** by **reducing** the **load** on **Docker's file monitoring system**.

7) Compose.yaml:

Path: cd danswer/deployment/docker_compose/compose.yaml

In our **development setup** for **Danswer**, I **began** with the **original compose**.**yaml file provided** in the **Danswer GitHub repository**. This **file** defines the **primary services required** for **running** the **application**, **including api_server**, **web_server**, **background**, **relational_db**, **index**, and others **necessary** for the **backend infrastructure**. The **initial compose**.**yaml configuration** was **meant** for **general development** and **testing** of **core services**.

8) Raised Queries on Docker Community & Danswer Community Github:

- **https://forums.docker.com/t/dockerized-web-server-not-reflecting-code-changes-despite-sync-confirmation-using-docker-compose-watch/144832**

- **https://github.com/danswer-ai/danswer/issues/3054**

To access the Danswer application as an admin, please log in as:
Email: noelshallum@gmail.com
Password: 12345

- **Ibrahim Sultan**