Reflection on Dockerizing the QR Code Generator

Introduction

As someone new to Docker and not primarily a developer, I initially needed time to understand the concept. After researching containerization, I was amazed by Docker's ability to package an application into a portable unit that can run anywhere with guaranteed consistency. This reflection document outlines my key experiences and challenges encountered while Dockerizing the QR Code Generator application, highlighting the steps I took, issues I faced, and lessons learned throughout the process.

Detailed Workflow Steps

Before diving into Dockerization specifics, here are the concrete steps I followed as I worked through the assignment:

1. **Cloned and Inspected Repository**
    a. Cloned https://github.com/Hanyyoussef4/Assignment7.git and opened it in VSCode.
    b. Reviewed the project structure: main.py, requirements.txt, Dockerfile, .github/workflows/ci.yml, plus qr_codes/, screenshots/, and tests/ folders.
2. **Verified Core Functionality**
    a. Ran python main.py --help to confirm the CLI interface.
    b. Confirmed that running without flags generated github_qr.png and docker_qr.png in qr_codes/ using default environment-variable values.
3. **Updated README and Screenshots**
    a. Added usage examples and the GitHub Actions status badge to readme.md.
    b. Generated container and workflow screenshots, saved them in screenshots/, and referenced them in the README.
4. **Dockerfile Creation and Iteration**

   a. Selected python:3.11-slim as the base image for a compact footprint.

   b. Copied requirements.txt first to leverage Docker layer caching, then installed Python dependencies.

   c. Encountered build errors due to missing system libraries; resolved by installing build-essential and libffi-dev before pip install.

5. **Docker Compose and Volume Mounts**

   a. Wrote docker-compose.yml to mount the host qr_codes/ directory for persistent storage.

   b. Tested docker-compose up to verify generated files persisted on the host filesystem.

6. **CI Workflow Development**

   a. Configured .github/workflows/ci.yml to build the Docker image, run pytest inside the container, and push to Docker Hub.

   b. Added caching for Docker layers and pip packages to speed up workflow runs.

   c. Resolved YAML syntax errors by validating against GitHub's schema and added DOCKERHUB_USERNAME and DOCKERHUB_TOKEN as secrets.

7. **Commit and Final Validation**

   a. Verified that docker run successfully generated QR codes within a fresh container.

   b. Ensured the GitHub Actions run passed all stages, and confirmed the Docker image was pushed to the registry.

1. Understanding the Application Requirements

- **Analyzed scope**: Inspected dependencies (requirements.txt), environment variables, and expected outputs.

- **Defined objectives**: Reliable container builds, seamless CLI behavior inside Docker, and smooth CI/CD integration.

2. Writing the Dockerfile

- **Base image choice**: python:3.11-slim for minimal size; added system packages as needed.
- **Caching strategy**: Separated dependency installation from code copy to reuse layers.

Challenges

- Missing build-essential and libffi-dev caused pip failures; learnt to pre-install system libs.
- Correct placement of WORKDIR /app to avoid file copy mishaps.

3. Configuring Environment Variables

- Exposed QR_CODE_DIR, FILL_COLOR, and BACK_COLOR in both the Dockerfile and docker-compose.yml.
- Standardized .env usage and updated documentation accordingly.

Challenges

- Harmonizing Docker Compose environment parsing with local .env behavior.

4. Lessons Learned

- Small base images reduce attack surface and speed up pulls.
- Layer caching dramatically improves build times in CI.
- Consistent environment-variable handling is vital across development and CI.

- Logging to stdout/stderr is key for diagnosing container issues.

5. Reflections from Other Projects

CLI Calculator (Assignment 5)

- Built GitHub Actions to run pytest, enforce 100% coverage, and cache dependencies.
- Overcame YAML schema errors and optimized caching.

Linux & Git Cheat-Sheet (Midterm Project)

- Automated deployment to GitHub Pages via workflow; integrated markdown lint and link checks.

QR Code Generator App (Assignment 7)

- Applied enhanced Docker layering, environment management, and CI secrets handling.

Conclusion

Containerizing and automating the QR Code Generator—and applying those patterns to past projects—strengthened my grasp of Docker best practices, CI/CD workflows, and environment management. The hands-on troubleshooting reinforced the value of clear documentation, cache optimization, and robust error handling in production-like environments.