BEAM GUI Dockerization — Technical Notes

Team BEAM

October 30, 2025

Contents

1 Prerequisites

- Install **Docker Desktop** (macOS).

2 Quickstart

```
git clone <repo-url>
cd <repo-folder> # project root
docker compose up -d
# open http://localhost:6080
# stop: docker compose down
```

3 Introduction

The objective is to containerize the BEAM GUI so it runs reliably on macOS (Apple Silicon) by hosting a lightweight Linux desktop inside Docker and exposing it in the browser via noVNC. The container bundles all native and Python dependencies (PyQt5, GNU Radio 3.8, `gr-osmosdr`, X11/Qt libraries), auto-starts the GUI on boot, and mounts the host source folder for fast iteration.

Goal

- Run the existing BEAM GUI (`main.py`) on macOS using Docker

Scope & Constraints

- Base image: Ubuntu 20.04 LXDE + noVNC (`dorowu/ubuntu-desktop-lxde-vnc:focal`).
- Platform: `linux/amd64` (runs under emulation on Apple Silicon; acceptable performance).
- Display server: virtual X (`Xvfb`) on `:1` with Qt/X11 runtime flags.

4 Docker Runtime Details

The container exposes a full desktop session over HTTP and launches the BEAM GUI automatically after the X display becomes available.

Operational Workflow (Step by Step)

1. **Install Docker Desktop (macOS).**
   Download and install Docker Desktop. Open it once so it starts running.
2. **Get the project folder.**
   Either clone the repo from git or unzip the folder so you have the files on your Mac.
3. **Open Terminal and go to the project folder.**

   ```
   cd <repo-folder> # the folder that contains docker-compose.yml
   ```

4. **Start it (first time builds, then runs).**
   The first time may take a few minutes because it downloads and builds everything. Type this on terminal so it builds:

   ```
   docker compose up -d
   ```

5. **Open the desktop in your browser.**
   In any browser on your Mac, go to:

   ```
   http://localhost:6080
   ```

   You should see a Linux desktop. The BEAM GUI window will appear automatically shortly after.

6. **(If nothing appears) quick check.**
   Make sure the container is running and the port is mapped.

   ```
   docker compose ps
   # Look for: "0.0.0.0:6080->80/tcp" and STATUS "Up"
   ```

7. **Stop when you are done.**

   ```
   docker compose down
   ```

8. **Past the first run you can simply start again:**

   ```
   docker compose up -d
   # open http://localhost:6080
   ```

5 Project Files Overview

This section summarizes the files required to build and run the BEAM GUI in Docker. The *project root* (the folder with the files below) is mounted into the container at `/opt/app`.

Quick Terms

- **Image** — the *bundle/blueprint*. A ready-made package with OS bits, libraries, and our code (built from the `Dockerfile`). You create containers from images.
- **Container** — the *running instance* of an image. In our setup the container is named `beam-gui`.
- **Volume (bind mount)** — a *shared folder* between your Mac and the container. Here, `./` is mounted at `/opt/app`, so edits on your Mac appear immediately inside the container.
- **Port mapping** — how your browser reaches the container. `6080:80` means "host port 6080" forwards to "container port 80," so the desktop is at `http://localhost:6080`.

File	Purpose
`docker-compose.yml`	Compose "runbook": builds the image, creates container `beam-gui`, maps ports (`6080:80`, `5901:5901`), mounts the project folder to `/opt/app`, sets `RESOLUTION`, and restarts policy.
`Dockerfile`	Builds the image from `dorowu/ubuntu-desktop-lxde-vnc:focal`; installs Python, PyQt5, GNU Radio, `gr-osmosdr`, and X11/Qt libraries; copies project files; wires the supervisor program.
`launch_app.sh`	Launcher invoked by `supervisord`. Waits for X display `:1`, sets Qt/X env (`DISPLAY=:1`, `QT_QPA_PLATFORM=xcb`, `XDG_RUNTIME_DIR`, etc.), then runs `python3 -u main.py`.
`app-supervisord.conf`	Supervisor program `beam_app`: `autostart=true`, `autorestart=true`, runs `/usr/local/bin/launch_app.sh`; logs to `/var/log/app_stdout.log` and `/var/log/app_stderr.log`.
`requirements.txt`	Optional Python dependencies installed at build time via `pip3`; keep minimal because most heavy deps are apt-based.
`main.py`	Application entry point(main code) for the BEAM GUI (PyQt5).
`sun_radiometer.py` (and other modules)	Application modules; include imports for GNU Radio (`gnuradio.qtgui`) and `osmosdr`.
`start.sh` (reference)	Earlier startup helper (kept for reference); after supervisor integration the app launches via `launch_app.sh`.
`.dockerignore` (optional)	Exclude large/irrelevant files (e.g., `*.exe`, `__pycache__/`, `.git/`) to speed builds and keep images small.

Note. The container's noVNC web UI is served on port **80** inside the container; the compose file maps host `6080` to container `80`, so the desktop is reached at `http://localhost:6080`.

docker-compose.yml — "How to run it" (one command)

Works as the remote control for running the app.

- **Build from here:** `build: .` — use the local `Dockerfile` to build the image.
- **Name:** `container_name: beam-gui` — the container will show up as "beam-gui".
- **Architecture:** `platform: linux/amd64` — runs this desktop image on Apple Silicon Macs.
- **Open in browser:** `ports: "6080:80"` — visit `http://localhost:6080` on your Mac; Docker forwards you to the container's web desktop (noVNC) on port **80**.
- **Optional VNC:** `5901:5901` — connect a native VNC client to `localhost:5901` if desired.
- **Live code mount:** `volumes: ./:/opt/app` — edits on your Mac appear instantly inside the container.
- **Desktop size:** `RESOLUTION=1280x800` — change for smoother performance if needed.

start.sh — "Start desktop, then app" (simple launcher)

This script starts the desktop processes, waits for the virtual screen to exist, sets a few Qt variables, and launches the GUI:

1. Starts the desktop stack via `supervisord`.
2. Waits until the X display socket `/tmp/.X11-unix/X1` exists (`DISPLAY = :1`).
3. Prepares a runtime folder for Qt WebEngine (`XDG_RUNTIME_DIR`).
4. Sets safe Qt flags for virtual X (`QT_QPA_PLATFORM=xcb`, `QT_X11_NO_MITSHM=1`, `LIBGL_ALWAYS_INDIRECT=1`
5. Runs `python3 -u main.py` from `/opt/app`.

Note: What these Qt variables do Inside Docker there is no real monitor, so the desktop uses a virtual screen (`Xvfb`) on `DISPLAY=:1`. These variables are just switches that make the GUI behave nicely there:

- `DISPLAY = :1` — points the app to the virtual screen created by Xvfb.
- `QT_QPA_PLATFORM = xcb` — tells Qt to use the standard X11 backend.
- `QT_X11_NO_MITSHM = 1` — disables a fragile shared–memory shortcut; more stable in virtual displays.
- `LIBGL_ALWAYS_INDIRECT = 1` — uses safe (indirect) OpenGL to avoid GPU/driver issues under Xvfb.
- `XDG_RUNTIME_DIR = /tmp/runtime-root` — a writable runtime folder modern Qt parts need.
- `QTWEBENGINE_DISABLE_SANDBOX = 1` — required when running as root in a container so QtWebEngine starts.
- `PYTHONUNBUFFERED = 1` — makes Python print logs immediately (handy for debugging).

launch_app.sh — "App-only launcher" (used by supervisord)

A small script that only waits for the X display `:1`, sets the same Qt variables, and executes `python3 -u main.py`. This is what `supervisord` runs automatically so your GUI appears every time the container starts.

Python sources (our actuall program)

`main.py` (GUI entry point), `sun_radiometer.py` (logic; imports GNU Radio and `osmosdr`), and any other `.py` modules are the actual application code. Docker doesn't change them; it just provides the correct Linux + libraries so they can run on macOS.

requirements.txt (optional Python packages)

If your app needs extra Python libraries from `pip`, list them here. They are installed at build time by the `Dockerfile`.

6 The Dockerfile (Short Explanation)

The `Dockerfile` builds one ready-to-run image (a bundle) that already contains Ubuntu + desktop, Python/PyQt5, radio libraries, and our small launch files. Workflow:

1. Start from an Ubuntu desktop image that already has the web desktop (noVNC).
2. Install system packages we need: Python 3, PyQt5, GNU Radio, `gr-osmosdr`, and basic X11/Qt libs.
3. (Optional) Install Python packages listed in `requirements.txt`.

4. Copy our project into `/opt/app` and add a tiny launcher so the GUI starts automatically.

5. Expose the web desktop port (80) — Compose maps it to `http://localhost:6080`.

Final Result: after this build, anyone with Docker can run the GUI with one command (`docker compose up -d`) — no manual installs, no setup on their machine.