# Running GUI apps within Docker containers

Typically Docker is used to encapsulate server-side software in reproducible packages - containers. A certain degree of isolation is ensured between containers. Furthermore, containers can be used as building blocks for systems consisting of multiple software servers. For example, a web app can consist of backend server, database server, frontend server, load balancer, redis instance for caching and so on.

However, what if we want to run desktop GUI apps within Docker containers to use them as components within larger systems? For example, if we run Firefox within Docker we can have an explicit separation of browser state between containers. This is beneficial for things like social media management, growth hacking (either via social media automation or manual labour done by VAs) or OSINT investigations. For example, one container would be configured with Firefox instance that uses a single dedicated mobile proxy for just one social media account. This would provide a degree of protection against social media platform cracking down on sock puppet accounts being used from single setup because traffic is kept separate for each account and cookie cross-contamination is being prevented.

Suppose we have Docker installed on macOS or other Unix/Linux system. How do we run Firefox within Docker container? For a first attempt, let us consider the following Dockerfile:

```
FROM ubuntu:latest
RUN apt-get update && apt-get install -y firefox x11vnc xvfb
RUN echo "exec firefox" > ~/.xinitrc && chmod +x ~/.xinitrc
CMD ["x11vnc", "-create", "-noxrecord", "-noxfixes", "-noxdamage", "-forever",
```

We use `ubuntu` base image and install Firefox with two more things:

- x11vnc - VNC server for remote access of X11-based desktop environments. Think of it as poor-mans TeamViewer.
- xvfb - a version of X11 server that does not require any actual display hardware and renders video into RAM instead.

We configure X11 to run Firefox at startup and launch x11vnc with the exact arguments that allow interoperability with macOS on the client side.

Building and running this container is rather simple:

```
$ docker build -t firefox-test-1 .
$ docker run -p 5900:5900 --rm firefox-test-1
```

We can use VNC feature in macOS Finder to access it by choosing "Go -> Connect to server", putting in `vnc://127.0.0.1:5900` into text field at the top and pressing Connect. It will ask for password - enter `trustno1`. This is a minor inconvenience we have to deal on macOS to make it work.

Screenshot 1

This is easy to get working, but relies on VNC capability on client side. It would be nice to avoid it. Assuming we have X11 server installed on the host system, let us remove x11vnc from container and install just the bare minimum of extra packages to make Firefox work in X11 client environment that will create an outgoing connection. This results in the following simplified Dockerfile:

```
FROM ubuntu:latest
RUN apt-get update && apt-get install -y libxext-dev libxrender-dev libxtst-de
CMD firefox
```

Let us put it into separate directory and build a container from it:

```
$ docker build -t firefox .
```

Now we need to set up X11 server with incoming connections enabled. On Linux you may need to just run `xhost +`, but on macOS you need to install XQuartz, launch it, go to Preferences and check the box that says "Allow connections from network clients", then run `xhost +` in terminal. Now we are ready to run the Firefox container that we have built before:

```
$ docker run --rm -e DISPLAY="$(ifconfig en0 | grep inet | awk '$1=="inet" {pr
```

Note that we set `DISPLAY` environment variable to a value that includes IP address of host system, and `0`, separated by colon. This tells the X11 client part within container to reach out to X11 server at that particular IP address and use zeroth display for new windows.

Screenshot 2

This does simplify environment within Docker container, but makes requirements for host machine more complicated, especially if it's a system that does not natively support X11. Furthermore, if we wanted to run this kind of setup on a server environment we would need to have desktop environment running there already, which has its own problems. There has to

be a better way.

Turns out, there is a web app called novnc that bridges the gap between x11vnc and web browser. It works as a client for x11vnc, but as a server to the end user that is connecting remotely via web browser. Although it does not have an official Dockerfile or Docker Hub image, there are some unofficial ones available. For this example, we will use theasp/novnc image from Docker Hub.

We keep the Firefox container as-is, but we create a docker-compose.yml file that brings both parts together (based on example from theasp/novnc Github repo):

```
version: '2'
services:
  firefox:
    image: firefox
    environment:
      - DISPLAY=novnc:0
    depends_on:
      - novnc
    networks:
      - x11
  novnc:
    image: theasp/novnc:latest
    environment:
      # Adjust to your screen size
      - DISPLAY_WIDTH=1600
      - DISPLAY_HEIGHT=968
      - RUN_XTERM=no
    ports:
      - "8080:8080"
    networks:
```

```
      - x11
networks:
  x11:
```

Running `docker-compose up` builds and launches both containers and we can access the Dockerized Firefox via regular web browser at: http://localhost:8080/vnc.html

Screenshot 3

With this setup, we have one static part (x11vnc + novnc container) and one part that we would replace if we wanted to run some other GUI program within Docker environment. Furthermore, we no longer require any client side software other than web browser. We can also run this kind of configuration on cloud deployment (e.g. Docker server from Digital Ocean marketplace) if we solved the security issue of no access control being enforced. These are significant improvements over our first attempt.

Let us also look into some open source projects dedicated to running GUI apps within Docker containers.

## x11docker

x11docker is a solution for running GUI apps within Linux host system that is more advanced in some ways (GPU access, peripheral support, security improvements), but is constrained to an existing Linux desktop environment and does not properly support operating systems other than Linux.

To launch a GUI app through x11docker, one would have to build or pull Docker container for that app written in a way that is compatible with x11docker (some examples are on Github and Docker Hub) and launch it

through x11docker CLI tool.

## Kasm Workspaces

Kasm Workspaces is an open source platform that takes this idea quite a bit further and even offers a SaaS version that they host for you. It is also available through Digital Ocean marketplace at: https://marketplace.digitalocean.com/apps/kasm-workspaces

Like x11docker, it provides a bunch of pre-developed Docker images for common apps like Firefox (and other browsers), Slack, Discord, VS Code and others. Entire Linux desktop environments are available through one-click install on management interface. Like with out previous example, remote desktops are made available through web browser. However Kasm Workspaces use KasmVNC - a VNC solution that is developed by the same company.

## docker-android

So far we talked about running Linux GUI apps inside containers. But what about running Android apps? Turn out, it is possible to run an entire Android emulator within Docker container and expose it through novnc interface by using solution called docker-android. Even though it warns about nested virtualization not being completely supported, I was able to run it through the following command on Digital Ocean VPS:

```
$ docker run --privileged -d -p 6080:6080 -p 5554:5554 -p 5555:5555 -e DEVICE=
```

Android emulator requires some CPU and RAM thus making it infeasible to run it on $5/month droplets - it simply was too slow. However the

performace was acceptable on some of the larger servers - it could even play Youtube videos without sound.

Screenshot 4

Since we also have port 5554 and 5555 exposed, we could also use ADB to side-load APK files and do some debugging.