Making a docker image to run the Waterfox browser

Neville Jackson

4 Sep 2022

1 Introduction

This project is about learning how to compose a Dockerfile which runs an X11 application. The Waterfox browser was chosen as an example X11 application, because Waterfox is not available as a package in most Linux distributions, so the result might actually be a useful means of obtaining Waterfox without having to download a tarfile and do a local install.

Getting a running Docker container to talk to the X11 server in the host system is relatively easy. Waterfox is a large and complex X application, and most of the challenge of this project is in setting up in the Docker image a working environmejnt for Waterfox.

2 What is involved in a normal Waterfox install?

There is a document [16] on that. Briefly the steps are

1. Download a tarfile from the Waterfox website [15] or from the Github site [17].
2. Unpack the tarfile in /usr/local/src

   ```
   bzip2 -dc waterfox-G4.1.1.1.en-US.linux-x86_64.tar.bz2 | tar xvf -
   ```

3. Make a link in /usr/local/bin to point to the Waterfox binary

   ```
   ln /usr/local/src/waterfox/waterfox /usr/local/bin/waterfox
   ```

4. Define the desktop icon by adding a file waterfox.desktop to the directory /usr/share applications. The file waterfox.desktop does not come with the tarfile, but is available here [16]

We need to implement steps 1 to 3 in a Dockerfile. Step 4 can be omitted, as there will be no icon if running Waterfox in a container. The container will have to be started from the command line.

3 How to run a docker container requiring access to the host Xserver

We assume the host is running X11 (not Wayland). Set up the following primitive Dockerfile.

```
FROM debian
WORKDIR /home/demo
RUN apt-get -y update
RUN apt-get -y upgrade
RUN apt-get -y x11-apps
RUN groupadd -g 1000 demo
RUN useradd -d /home/demo -s /bin/bash -m demo -u 1000 -g 1000
USER demo
ENV HOME /home/demo
CMD /usr/bin/xcalc
```

Build this with

```
docker build -t nevj/test:v2 .
Sending build context to Docker daemon  2.048kB
Step 1/10 : FROM debian
 ---> 07d9246c53a6
Step 2/10 : WORKDIR /home/demo
 ---> Running in 2e0e40e0fd4c
..........
Successfully built 517c1937106a
Successfully tagged nevj/test:v2
```

Then run it with

```
[nevj@trinity dtest]$ xhost +
[nevj@trinity dtest]$ docker run -v /tmp/.X11-unix:/tmp/.X11-unix \
          -e DISPLAY=$DISPLAY -h $HOSTNAME \
          -v $HOME/.Xauthority:/home/nevj/.Xauthority nevj/test:v2
```

and the xcalc app appears in a separate X window, as shown in Figure 1

The *xhost +* statement is needed to give the container permission to communicate with the host X11 server. The purpose of the options included in the *docker run* statement are as follows

-v /tmp/.X11-unix:/tmp/.X11-unix defines a *mount* for a volume. The string */tmp/.X11-unix:/tmp/.X11-unix* is the mount point on the host. The part before *':'* is the volume name on the host. The part after *':'* is where the file or directory is mounted oinside the container. A volume is just a file or directory in the host filesystem.

-e DISPLAY=$DISPLAY sets the environment variable DISPLAY in the container

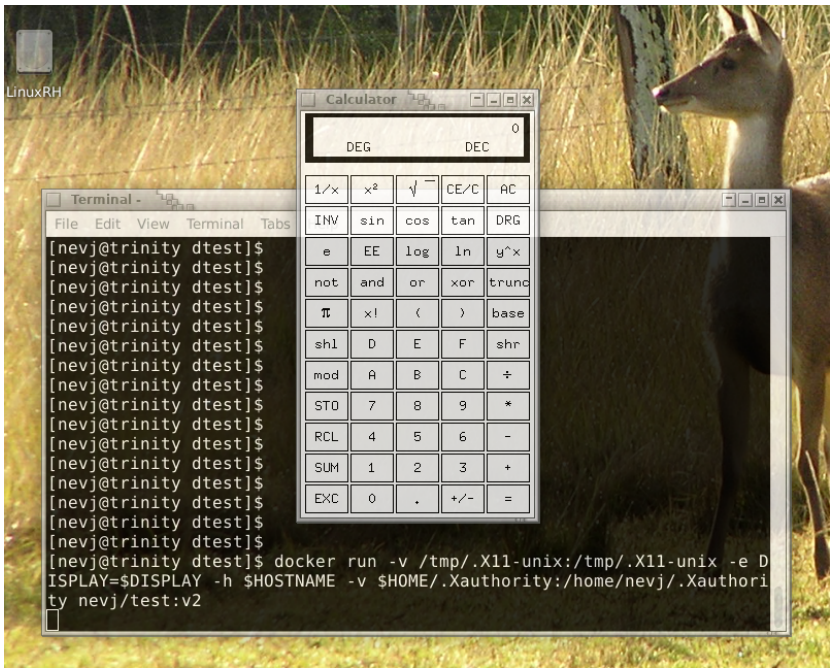

Figure 1: Xcalc app started from a Docker container

-h $HOSTNAME sets the container hostname

$HOME/.Xauthority:/home/nevj/.Xauthority sets the .Xauthority file of the container to the .Xauthority file of the host user who initiated the *run* statement again using a volume mount.

nevj/test:v2 is the name of te image, build as above.

Volumes are the mechanism used by Docker containers to share data between the host and the container (and between containers). They are need in this case because the container is an X11 client, while the host acts as the X11 server, so information has to pass between container and host. A docker container can only write on the part of the host filesystem defined as a docker volume. There is also *tmpfs mount* which is a temporary ram disk used for non-permanent data within a container.

4 First attempt at a Waterfox Dockerfile, using a Debian parent image

There is no new principle involved in running Waterfox rather than the simple xcalc app, in a docker container. The Waterfox browser is a large program with

many dependencies, so the Dockerfile will have to build up a complete working environment specifically for Waterfox.

I chose Debian parent image for a first attempt, because Debian is familiar and is most likely to compatable with Waterfox. The Debian parent image is large. We shall try later to repeat the job with a smaller parent image.

4.1 Finding the dependencies of a binary program file

There is a really good article [14] on identifying dependencies of a runtime binary when making a dockerfile.

If one looks at the unpacked Waterfox ditribution tarfile, it contains the files shon in Figure 2

Figure 2: Files contained in the waterfox distribution tarfile

The *.so files shown in green are dynamically loaded libraries. Presumably these are all dependencies of waterfox, but they may not be the only dependencies. There is a file TwemojiMozilla.ttf in the fonts subdirectory, and there are various default configuration files.

An initial check for dependencies can be obtained with *ldd*

```
nevj@mary:/usr/local/src/waterfox$ ldd waterfox
linux-vdso.so.1 (0x00007ffc3eb95000)
libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f2a6d46c000)
libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f2a6d466000)
libstdc++.so.6 => /lib/x86_64-linux-gnu/libstdc++.so.6 (0x00007f2a6d299000)
libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007f2a6d155000)
libgcc_s.so.1 => /lib/x86_64-linux-gnu/libgcc_s.so.1 (0x00007f2a6d13b000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f2a6cf76000)
/lib64/ld-linux-x86-64.so.2 (0x00007f2a6d569000)
```

This is best done in a Debian system in which waterfox has been installed There are 8 libraries called directly by waterfox, and none of them are present in the distribution tarfile. These are libraries required for waterfox to start up. Note that linux-vdso.so.1 is a virtual library, ie it is built into the kernel and does not exist as a package. The other 7 should each be supplied by a package. We shall leave the problem of finding the package names which supply each library later.

There may also be libraries which waterfoox loads dynamically while executing. To find these we do the following, again in a Debian system

```
nevj@mary:~$ LD_DEBUG=libs waterfox >&logfile

nevj@mary:~$ cat logfile
     2572:     find library=libpthread.so.0 [0]; searching
      2572:      search cache=/etc/ld.so.cache
      2572:       trying file=/lib/x86_64-linux-gnu/libpthread.so.0
      2572:
      2572:     find library=libdl.so.2 [0]; searching
      2572:      search cache=/etc/ld.so.cache
      2572:       trying file=/lib/x86_64-linux-gnu/libdl.so.2
      2572:
      2572:     find library=libstdc++.so.6 [0]; searching
      2572:      search cache=/etc/ld.so.cache
      2572:       trying file=/lib/x86_64-linux-gnu/libstdc++.so.6
      2572:
      2572:     find library=libm.so.6 [0]; searching
      2572:      search cache=/etc/ld.so.cache
      2572:       trying file=/lib/x86_64-linux-gnu/libm.so.6
      2572:
      2572:     find library=libgcc_s.so.1 [0]; searching
      2572:      search cache=/etc/ld.so.cache
      2572:       trying file=/lib/x86_64-linux-gnu/libgcc_s.so.1
     2572:
      2572:     find library=libc.so.6 [0]; searching
      2572:      search cache=/etc/ld.so.cache
      2572:       trying file=/lib/x86_64-linux-gnu/libc.so.6
      2572:
      2572:
      2572:     calling init: /lib/x86_64-linux-gnu/libpthread.so.0
      2572:
      2572:
      2572:     calling init: /lib/x86_64-linux-gnu/libc.so.6
      2572:
      2572:
      2572:     calling init: /lib/x86_64-linux-gnu/libgcc_s.so.1
      2572:
      2572:
      2572:     calling init: /lib/x86_64-linux-gnu/libm.so.6
      2572:
      2572:
      2572:     calling init: /lib/x86_64-linux-gnu/libstdc++.so.6
      2572:
      2572:
```

```
     2572:     calling init: /lib/x86_64-linux-gnu/libdl.so.2
     2572:
     2572:
     2572:     initialize program: waterfox
     2572:
     2572:
     2572:     transferring control: waterfox
     2572:
     2572:
     2572:     calling init: /usr/local/src/waterfox/libnspr4.so
     2572:
     2572:
     2572:     calling init: /usr/local/src/waterfox/libplc4.so
     2572:
     2572:
     2572:     calling init: /usr/local/src/waterfox/libplds4.so
     2572:
     2572:     find library=librt.so.1 [0]; searching
     2572:      search cache=/etc/ld.so.cache
     2572:       trying file=/lib/x86_64-linux-gnu/librt.so.1
     2572:
     2572:
     2572:     calling init: /lib/x86_64-linux-gnu/librt.so.1
     2572:
     2572:
     2572:     calling init: /usr/local/src/waterfox/libmozsandbox.so
     2572:
     2572:
     2572:     calling init: /usr/local/src/waterfox/liblgpllibs.so
     2572:
     2572:
     2572:     calling init: /usr/local/src/waterfox/libnssutil3.so
     2572:
     2572:
     2572:     calling init: /usr/local/src/waterfox/libnss3.so
     2572:
 ...................
    2796:     calling init: /lib/x86_64-linux-gnu/libX11-xcb.so.1
     2796:
     2796:
     2796:     calling init: /usr/local/src/waterfox/libxul.so
     2796:
     2796:     /usr/local/src/waterfox/waterfox: error: symbol lookup error:
undefined symbol: nspr_use_zone_allocator (fatal)
     2796:
     2796:     calling init: /usr/local/src/waterfox/libsoftokn3.so
```

```
      2796:
      2796:
      2796:     calling init: /usr/local/src/waterfox/libfreeblpriv3.so
      2796:
JavaScript error: resource:///gre/modules/PartitioningExceptionListService.jsm
, line 69: NS_ERROR_MALFORMED_URI: Component returned failure code: 0x804b000
a (NS_ERROR_MALFORMED_URI) [nsIPartitioningExceptionListObserver.onExceptionL
istUpdate]
console.error: PushService:
  clearOriginData: Error clearing origin data:
  TypeError
JavaScript error: resource:///modules/Interactions.jsm, line 209: NS_ERROR_FA
ILURE: Component returned failure code: 0x80004005 (NS_ERROR_FAILURE) [nsIUse
rIdleService.removeIdleObserver]

###!!! [Parent][RunMessage] Error: Channel closing: too late to send/recv, me
ssages will be lost
```

Lots of messages, had to truncate it, was 5744 lines. The first 3 lines say it is searching for a system library called *libpthread*. The lines beginning *calling init* indicate the library has been found and give its absolute path. Some libraries may not be found, and they will have no line beginning *calling init*. We can get a list of libraries found with

```
nevj@mary:~$  cat out | grep init > outinit
nevj@mary:~$ wc outinit
  715  2878 45875 outinit
nevj@mary:~$  cat outinit
   2572: calling init: /lib/x86_64-linux-gnu/libpthread.so.0
      2572: calling init: /lib/x86_64-linux-gnu/libc.so.6
      2572: calling init: /lib/x86_64-linux-gnu/libgcc_s.so.1
      2572: calling init: /lib/x86_64-linux-gnu/libm.so.6
      2572: calling init: /lib/x86_64-linux-gnu/libstdc++.so.6
      2572: calling init: /lib/x86_64-linux-gnu/libdl.so.2
      2572: initialize program: waterfox
      2572: calling init: /usr/local/src/waterfox/libnspr4.so
      2572: calling init: /usr/local/src/waterfox/libplc4.so
      2572: calling init: /usr/local/src/waterfox/libplds4.so
      2572: calling init: /lib/x86_64-linux-gnu/librt.so.1
      2572: calling init: /usr/local/src/waterfox/libmozsandbox.so
      2572: calling init: /usr/local/src/waterfox/liblgpllibs.so
      2572: calling init: /usr/local/src/waterfox/libnssutil3.so
      2572: calling init: /usr/local/src/waterfox/libnss3.so
      2572: calling init: /usr/local/src/waterfox/libsmime3.so
      2572: calling init: /usr/local/src/waterfox/libmozsqlite3.so
      2572: calling init: /usr/local/src/waterfox/libssl3.so
      2572: calling init: /lib/x86_64-linux-gnu/libgpg-error.so.0
```

. . . . . .

So there are still 715 of them. The first 6 are the startiup dependencies, and the rest are runtime dependencies

There is also the question of configuration files.

References

[1] Docker tutorial. URL https://www.guru99.com/docker-tutorial.html

[2] Docker get started URL https://docs.docker.com/get-started/

[3] Docker Desktop URL https://docs.docker.com/desktop/install/linux-install/

[4] Docker Hub URL https://hub.docker.com/

[5] Official Dockerfile documnet URL https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

[6] Dockerfile Guide URL https://medium.com/@BeNitinAgarwal/best-practices-for-working-with-dockerfil es-fb2d22b78186

[7] Docker Basics: How to use Dockerfiles URL https://thenewstack.io/docker-basics-how-to-use-dockerfiles/

[8] A Beginners Guide to Understanding and Building Docker Images URL https://jfrog.com/knowledge-base/a-beginners-guide-to-understanding-and-building-docker-images/

[9] Best practices for writing Dockerfiles URL https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

[10] Creating a Docke Image for your Application URL https://www.stereolabs.com/docs/docker/creating-your-image/

[11] Docker Containerization Cookbook" - Hot Recipes for Docker Automation URL https://distrowatch.tradepub.com/free/w_java39/prgm.cgi?a=1

[12] Void Linux Docker Images URL https://github.com/void-linux/void-docker

[13] LibreWolf source code website. URL https://gitlab.com/librewolf-community/browser/source

[14] Rehn, A. (2021) Identifying application runtime dependencies: A toolkit for identifying the runtime libraries and associated data that applications require in order to run correctly inside containers. URL https://unrealcontainers.com/blog/identifying-application-runtime-dependencies/

[15] Waterfox website. URL https://www.waterfox.net

[16] Install the Waterfox browser on a Linux system. URL https://github.com/nevillejackson/Unix/blob/main/waterfox/waterfox.pdf

[17] WaterfocCo Github site URL https://github.com/WaterfoxCo/Waterfox/releases