**About Alpine Linux**

Alpine Linux is a lightweight distribution that is popular for deploying containerized apps because the virtual size of an Alpine Docker image starts at 5 MB. The virtual size of images based on other distributions exceeds 100 MB before adding anything else to them, so Alpine achieves a significant reduction in size.

Some key differences between Alpine and other distributions that are relevant to my project are:

* Alpine uses *musl lib* as a lightweight alternative to *glibc*. That means precompiled installers built on platforms with *glibc* could be useless, even if I install *libc6-compat* or *glibc* packages for Alpine Linux.
* Alpine drops old versions of packages when new versions become available, making version pinning impossible, so I probably won’t be able to use nodejs and npm packages from the official repositories to build Shiny Server.

Because of those characteristics, I assume that building the images will require compiling most key dependencies from source. I’ve never done that before, so working on this project will be a great (and painful) learning experience.

**Build from Native Packages**

Building from source can often take up a lot of time, so I decided to start with the native packages to build a working image as soon as possible. Here is what my Dockerfile looked like:

FROM alpine:3.8

# Declare environment variables ------------------

ENV LC\_ALL en\_US.UTF-8

ENV LANG en\_US.UTF-8

ENV BUILD\_DEPS \

cairo-dev \

libxmu-dev \

openjdk8-jre-base \

pango-dev \

perl \

tiff-dev \

tk-dev

ENV PERSISTENT\_DEPS \

R-mathlib \

gcc \

gfortran \

icu-dev \

libjpeg-turbo \

libpng-dev \

make \

openblas-dev \

pcre-dev \

readline-dev \

xz-dev \

zlib-dev \

bzip2-dev \

curl-dev

# Install R and R-dev ------------------

RUN apk upgrade --update && \

apk add --no-cache --virtual .build-deps $BUILD\_DEPS && \

apk add --no-cache --virtual .persistent-deps $PERSISTENT\_DEPS && \

apk add --no-cache R R-dev && \

apk del .build-deps

CMD ["R", "--no-save"]

The resulting image has a virtual size of 136.4 MB. That’s an improvement in size compared to the rocker/r-base image, which is 277.4MB according to MicroBadger. However, I should mention that the rocker/r-base image includes some packages like *litter* that my Dockerfile doesn’t, so it has additional features that could be of interest to R users. (Not to mention that it is also more stable for use in production than my image.)

**Build from Source Code**

I recently learned that Alpine drops old versions of packages from the repository when new versions become available . That means some versions of R won’t always be available from the official repositories, so I decided to create a Dockerfile for building R from source as well. I made the following changes to the previous Dockerfile:

* Declared environment variables for storing the R version and target directory for the source code.
* Added *wget* and *tar* to build dependencies for getting and unpacking the sources.
* Added *libint* and *g++* to runtime dependencies. I learned that *g++* is necessary for building from source after the first attempt raised an error while setting up the configuration options. The first container I started failed to run R because package *libint* was not installed, so I added it later.
* Removed the *R-mathlib* package from runtime dependencies. Rmath is installed as a standalone library from source in this image.
* Finally, I changed the RUN command accordingly to get, configure, and install R from source instead of adding the native packages.

This is what the final version of the file looked like:

FROM alpine:3.8

ENV LC\_ALL en\_US.UTF-8

ENV LANG en\_US.UTF-8

ENV R\_VERSION 3.5.1

ENV R\_SOURCE /usr/local/src/R

ENV BUILD\_DEPS \

cairo-dev \

libxmu-dev \

openjdk8-jre-base \

pango-dev \

perl \

tiff-dev \

tk-dev \

wget \

tar

ENV PERSISTENT\_DEPS \

libint \

gcc \

g++ \

gfortran \

icu-dev \

libjpeg-turbo \

libpng-dev \

make \

openblas-dev \

pcre-dev \

readline-dev \

xz-dev \

zlib-dev \

bzip2-dev \

curl-dev

RUN apk upgrade --update && \

apk add --no-cache --virtual .build-deps $BUILD\_DEPS && \

apk add --no-cache --virtual .persistent-deps $PERSISTENT\_DEPS && \

mkdir -p $R\_SOURCE && cd $R\_SOURCE && \

wget https://cran.r-project.org/src/base/R-3/R-${R\_VERSION}.tar.gz && \

tar -zxvf R-${R\_VERSION}.tar.gz && \

cd R-${R\_VERSION} && \

./configure --prefix=/usr/local \

--without-x && \

make && make install && \

cd src/nmath/standalone && make && make install && \

apk del .build-deps

CMD ["R"]

The virtual size of the image it built was 311.6 MB, but it should be possible to reduce the image even further by cleaning up after building R. I didn’t want to do that yet because I still had to run the container and perform make check from the shell to test the installation.

There was one non-fatal error (raised by shownNonASCIIfile()) and two fatal errors raised by timezone.R and reg-tests-1c.R tests:

comparing ‘tools-Ex.Rout’ to ‘tools-Ex.Rout.save’ ... NOTE

--- /tmp/RtmpdpmEgM/Rdiffa445b3d67023

+++ /tmp/RtmpdpmEgM/Rdiffb445b7ac41733

@@ -796,8 +796,8 @@

> cat(out, file = f, sep = "\n")

>

> showNonASCIIfile(f)

-1: fa\*ile test of showNonASCII():

-4: This has an \*mlaut in it.

+1: faile test of showNonASCII():

+4: This has an <fc>mlaut in it.

> unlink(f)

>

>

.

.

.

running code in 'timezone.R' ...make[4]: \*\*\* [Makefile.common:105: timezone.Rout] Error 1

make[4]: Leaving directory '/usr/local/src/R/R-3.5.1/tests'

Sys.timezone() appears unknown

.

.

.

running code in 'reg-tests-1c.R' ...make[3]: \*\*\* [Makefile.common:105: reg-tests-1c.Rout] Error 1

make[3]: Leaving directory '/usr/local/src/R/R-3.5.1/tests'

make[2]: \*\*\* [Makefile.common:291: test-Reg] Error 2

make[2]: Leaving directory '/usr/local/src/R/R-3.5.1/tests'

make[1]: \*\*\* [Makefile.common:170: test-all-basics] Error 1

make[1]: Leaving directory '/usr/local/src/R/R-3.5.1/tests'

make: \*\*\* [Makefile:240: check] Error 2