Using a container to provide 32 bit gcc5 compilation services on a 64bit Linux system

Bill Stoddart

June 30, 2024

Abstract

At EuroForth 2023 we mentioned that our reversible Forth had become impossible to maintain because the 32 bit gcc compiler is no longer supported. It was suggested that "containers" might provide a solution. Here we report on where that suggestion has led us.

1 Introduction

According to Red Hat, "A Linux container is a set of processes that are isolated from the rest of the system. All the files necessary to run them are provided from a distinct image."

This definition is fine so long as we don't take the term "isolation" too seriously. We will be using a 32 bit linux running in a container to provide 32 bit gcc compilation services required by the 64 bit Linux host system. The source files to be compiled and the resulting binary files will be on the 64 bit host, but all compilation will be done within the container. This will be possible because we are able to mount a directory of the host into the file system of the Linux running in the container. Apart from this the processes in the container are isolated from the host system.

Specific technologies used are the Docker package for containers, and 64 bit Fedorer 38 as the host. The name of our reversible Forth is `RVM_FORTH`.

2 Overview

2.1 Building our container image

We did not have to create a container from scratch. Docker provides an archive of pre-built container images, along with a search facility, and in the repository we found the entry:

```
frankwolf/32bit-ubuntu                32bit Ubuntu docker image
```

We pulled down this image from the reposotory and ran it in a container. In the current context we run the 32 bit Linux image and set it executing `bash`. At this point the prompts in the terminal window change so we can see we are running Linux in the container.

Containers are often describes as lightweight virtual machines, and this is the impression we have at this point. We are working on a 32 bit Ubuntu in a terminal window, but on a 64 bit host.

The Frankwolf image did not include support for gcc[1]. To provide this we use the command line to install the additional packages required for compilation of our code, e.g. `gcc5, g++5, binutils` and `make`.

When this is complete we exit from the container by entering the command `exit` and post the updated image back to the repository as

`billstoddart/ubuntu32bit_gcc5:vsn0`

2.2 The Forth "home" directory.

Our `RVM_FORTH` uses the concept of a "home" directory. During meta compilation Forth source code and C packages invoked from the Forth nucleus are located relative to this home directory. Later, when Forth runs, additional packages containing definitions that are not included in the nucleus are also located relative to the home directory.

The way in which a newly minted Forth gets to know where its home directory is necessarily different when we build it on a container, and we will come to these details in due course.

2.3 Compilation in the container

We use a bash script containing a Docker `run` command to pull down our saved container image from the Docker repository and mount the Forth home directory (which in our case is located at `/home/bill/rvm/rvm`) at `/root/rvm` in the filesystem of the 32 bit Linux provided by the container. However, note that due to the expansion of a symbolic link the former appears as `/home/bill/Dropbox/bill/rvm/rvm/` in some scripts.

When we run a container we are able to give it a command to execute. In our case we set the container to run the bash script that compiles our Forth system. When this script terminates control returns to the Bash script running on the host.

[1] It has since been updated.

2.4 Installation of the Forth system

The binary produced by the above process is left in the home directory. It is executable as a Forth system, but is not yet findable in the search path for executables, and does not know where its home directory is. This is fixed by creating a script that invokes Forth and tells it where its home directroy is. This script is then moved into a suitable directory in the search path. Again, details will follow.

3 A closer look

3.1 Running our container, the details

To perform a meta comilation of the system we navigate to the Forth home directory on the host machine and invoke the script `./dmcomp` ("docker meta compile"). The script requires root privlages to run docker, and since the Forth home directory is not in the search path the invocation is via the command:

```
sudo ./dmcomp
```

The `dmcomp` script ensures that the Docker demon is running and removes any present containers which might otherwise interfere with running the current container. It then sets our container running, pulling the image from the docker archive if it is not present locally. with no-network connection and directs it to run `bash` on the script `/root/rvm/dcomp0` in the container.

Here is the contents of the `dmcomp` script.

```
#! /bin/bash
echo "Checking if docker demon is active"
FRED=$(systemctl is-active docker)
echo "response is"
echo $FRED
if
  [ "$FRED" != "active" ]
then
   echo "issuing command: systemctl start docker"
   systemctl start docker
fi
echo "Removing any present containers"
docker container prune -f
echo "Attempting to run bash in container: billstoddart/ubuntu32bit_gcc5:vsn0"
docker run -it --network none \
--name gcc5 \
-v /home/bill/rvm/rvm:/root/rvm \
billstoddart/ubuntu32bit_gcc5:vsn0 \
bash -c /root/rvm/dmcomp0
echo "Now we are back on the host in directory $(pwd)"
```

Within the `run` command we invoke a number of switches which have the following effects:

- `-it` Specifies that the container should have an interactve terminal. This allows us to report progress and interact with the container if an error condition arises.
- `--network none` Specifies that the container should have no network connections.
- `-v ...` Specifies that the Forth home directory should be mounted at `/root/rvm` in the container file system.

Finally the line `bash -c /root/rvm/dmcomp0` Specifies the container should execute `bash` on the script `/root/rvm/dmcomp0`. This is the script `dcomp0` in the Forth home directory, now found at `/root/rvm/dmcomp0` in the container's file system.

3.2 Running 32 bit compilation within the container.

The script `dmcomp0` invoked as described above contains the instructions for metacompiling the Forth system and compiling associated gcc libraries. The only point to note is that it must navigate to the Forth home directory on the container Linux, since it has not been invoked from that directory. So the script `dmcomp0` begins as follows:

```
cd /root/rvm #Navigate to Forth home directory
#A script to build RVM_FORTH
 ...
```

When the script terminates so does the Docker `run` command invoked in `dmcomp` and control returns to that script which is running on the host machine..

3.3 Distribution and installation of RVM_FORTH.

When Forth is built on a container it does not get to know where its home directory is.

In our case the binary executable `4TH_BIN` is given this information by being run in the script `RVM_FORTH` with the command:

```
4TH_BIN SET-HOME /home/bill/Dropbox/bill/rvm/rvm/ $@
```

When `4TH_BIN` executes it runs the Forth interpreter on the following text, which provides commands to set the `RVM_HOME` directory.

For a different user, the home directory will be in a different place. We provide an `install` script which is run just once when `RVM_FORTH` is first installed. Typical usage is:

```
sudo ./install /usr/bin
```

which would create the appropriate `RVM_FORTH` script and move it to `/usr/bin`

The contents of the `install` script are as follows:

```
#! /bin/bash
#Run this script when RVM Forth is first installed.
#It writes the script RVM_FORTH which invokes Forth via the binary
#4TH_BIN and sets its home directory. It places the script in the
#directory given by $1
#A typical invocation could be:   install /usr/bin
#It does not need to be re-run each time Forth is meta-compiled.
#echo "STARTUP=$(pwd)/4TH_BIN SET-HOME $(pwd)/ \$@" "
STARTUP="$(pwd)/4TH_BIN SET-HOME $(pwd)/ \$@"
echo "STARTUP = $STARTUP"
echo "$STARTUP > RVM_FORTH"
echo $STARTUP > RVM_FORTH
echo "Move the newly created script to the given directory"
echo "mv RVM_FORTH " $1
mv RVM_FORTH $1
echo "set execute permissions"
echo "chmod a+x" $1"/RVM_FORTH"
chmod a+x $1/RVM_FORTH
```

4 Problems

We were initially working on a Ubuntu host, but the version of Docker installed by the Ubuntu package manager had a bug which made it impossible to mount a directory from the host within the container Linux file system. The workaround was to provide the Forth system to the container Linux in the form of a tarball. We found this too clumsy to be our preferred solution. Attempting to uninstall Docker and install the version provided on the Docker website broke the package manager. We resolved the problem by switching to Docker on Fedorer.

5 Conclusions

Containers are an interesting technology that can ease problems of software instability on continually evolving systems. In our application they enabled us to use a 32 bit gcc compiler which is no longer supported in current versions of Linux.

Acknowledgement

Warm thanks to Gerald Wodni for taking an interest in our system maintainence problems and pointing us in the direction of containers.