

Building the Cross-Compilation Toolchain

Pieter P

To compile software for the Raspberry Pi, you need a cross-compilation toolchain. It's a collection of files and programs that you can run on your computer, and that produce a binary that can be executed on the Raspberry Pi.

Docker

I don't like to do a lot of installations on my main Linux box, so I build most of it inside of a Docker container. This has the added benefit that all builds are reproducible, and it's easy to undo the previous step or even start from scratch.

Installation

You need Docker and Docker Compose to build the toolchain. You can find detailed installation instructions here:

- [DigitalOcean - How to Install and Use Docker](#)
- [DigitalOcean - How to Install Docker Compose](#)

Dockerfiles

The actual Dockerfiles are on my GitHub, I'll briefly go over them on this page.

<https://github.com/tttapa/RPi-Cpp-Toolchain/tree/master/toolchain/docker>

Crosstool-NG

The toolchain is built using [Crosstool-NG](#).

It is installed in a CentOS 7 Docker container, because CentOS 7 was the oldest OS that I had to run the toolchain on. In this context, oldest refers to the Linux kernel version and the glibc version. They are backwards compatible, so you can run software compiled for an old version on a computer with a newer version, but you can't run software compiled for a new version on a computer with an older version.

The following Dockerfile downloads, builds and installs Crosstool-NG to the `~/local` directory of the container.

```
1 FROM centos:7 as ct-ng
2
3 # Install dependencies to build toolchain
4 RUN yum -y update && \
5     yum install -y epel-release && \
6     yum install -y autoconf gperf bison file flex texinfo help2man gcc-c++ \
7     libtool make patch ncurses-devel python36-devel perl-Thread-Queue bzip2 \
8     git wget which xz unzip && \
9     yum clean all
10
11 # Add a user called `develop` and add him to the sudo group
12 RUN useradd -m develop && echo "develop:develop" | chpasswd && usermod -aG wheel develop
13
14 USER develop
15 WORKDIR /home/develop
16
17 ENV CT_NG_VERSION=1.24.0
18 RUN git clone -b crosstool-ng-${CT_NG_VERSION} --single-branch --depth 1 \
19     https://github.com/crosstool-ng/crosstool-ng.git
20 WORKDIR /home/develop/crosstool-ng
21 RUN ./bootstrap && \
22     ./configure --prefix=/home/develop/.local && \
23     make -j$((nproc * 2)) && \
24     make install && \
25     cd && rm -rf crosstool-ng
26 ENV PATH=/home/develop/.local/bin:$PATH
```

The list of dependencies can be found on Crosstool-NG's GitHub: <https://github.com/crosstool-ng/crosstool-ng/blob/master/testing/docker/centos7/Dockerfile>

Raspberry Pi 3B+ 64-bit Cross-Compilation Toolchain

The following Dockerfile builds the toolchain for a Raspberry Pi 3B+ running a 64-bit operating system.

```

1 FROM crosstool-ng:v1 as aarch64-toolchain
2
3 WORKDIR /home/develop
4 RUN mkdir /home/develop/RPi3
5 WORKDIR /home/develop/RPi3
6 RUN ct-ng aarch64-rpi3-linux-gnu
7 RUN sed -i 's/CT_GCC_VERSION="[0-9.]*"/CT_GCC_VERSION="9.1.0"/g' .config && \
8     sed -i 's/CT_LINUX_VERSION="[0-9.]*"/CT_LINUX_VERSION="4.15"/g' .config && \
9     sed -i 's/CT_GLIBC_VERSION="[0-9.]*"/CT_GLIBC_VERSION="2.27"/g' .config
10 RUN ct-ng build && rm -rf .build
11
12 ENV TOOLCHAIN_PATH=/home/develop/x-tools/aarch64-rpi3-linux-gnu
13 ENV PATH=${TOOLCHAIN_PATH}/bin:$PATH
14 WORKDIR /home/develop

```

The configuration is based on the **aarch64-rpi3-linux-gnu** sample that comes with Crosstool-NG: <https://github.com/crosstool-ng/crosstool-ng/tree/master/samples/aarch64-rpi3-linux-gnu>

I changed the GCC version to the latest stable one, and the Linux kernel and glibc versions to match the versions that the Ubuntu Server 18.04 image ships with.

The build took around 25 minutes on a 2018 Dell XPS i7-8750H, and 45 minutes on a 2017 Dell XPS i7-7700HQ.

Raspberry Pi 3B+ 32-bit Cross-Compilation Toolchain

Even though the Raspberry Pi 3B+ has a 64-bit ARMv8/AArch64 CPU, at the time of writing, the Raspbian distribution is still only 32 bits. To be able to cross-compile software for this platform, you'll need a 32-bit cross-compiler.

```

1 FROM crosstool-ng:v1 as aarch32-toolchain
2
3 WORKDIR /home/develop
4 RUN sed -i 's/CT_ARCH_FPU="neon-vfpv4"/CT_ARCH_FPU="neon-fp-armv8"/g' \
5     .local/share/crosstool-ng/samples/armv8-rpi3-linux-gnueabi/crosstool.config
6 RUN mkdir /home/develop/RPi3
7 WORKDIR /home/develop/RPi3
8 RUN ct-ng armv8-rpi3-linux-gnueabi
9 RUN sed -i 's/CT_GCC_VERSION="[0-9.]*"/CT_GCC_VERSION="9.1.0"/g' .config && \
10     sed -i 's/CT_LINUX_VERSION="[0-9.]*"/CT_LINUX_VERSION="4.14"/g' .config && \
11     sed -i 's/CT_GLIBC_VERSION="[0-9.]*"/CT_GLIBC_VERSION="2.24"/g' .config
12 RUN ct-ng build && rm -rf .build
13
14 ENV TOOLCHAIN_PATH=/home/develop/x-tools/armv8-rpi3-linux-gnueabi
15 ENV PATH=${TOOLCHAIN_PATH}/bin:$PATH
16 WORKDIR /home/develop

```

The configuration is based on the **armv8-rpi3-linux-gnueabi** sample that comes with Crosstool-NG: <https://github.com/crosstool-ng/crosstool-ng/tree/master/samples/armv8-rpi3-linux-gnueabi>

I changed the GCC version to the latest stable one, and the Linux kernel and glibc versions to match the versions that the Raspbian Stretch image ships with.

Even though the CPU will be running in 32-bit mode, you can still use the ARMv8 NEON instructions, so I changed the compiler's default FPU flag to **neon-fp-armv8**. I haven't tested if it actually makes any difference.

The build took around 25 minutes on a 2018 Dell XPS i7-8750H, and 45 minutes on a 2017 Dell XPS i7-7700HQ.