Building the Cross-Compilation Toolchain

Pieter P

To compile software for the Raspberry Pi, you need a cross-compilation toolchain. A cross-compilation toolchain is a collection of development files and programs that you can run on your computer or workstation, that produce binaries that can be executed on a different system with a possibly different architecture, such as a Raspberry Pi.

Downloading a pre-built toolchain

If you need the cross-compiled programs and libraries like Python and OpenCV, the toolchain will be included in the next step.

If, on the other hand, you just want the toolchain, without any of the additional programs and libraries, you can install the toolchain separately. The easiest way is to just download and extract the pre-built toolchains from GitHub. Pick the one for the Raspberry Pi you need:

```
$ mkdir -p ~/opt
$ wget -qO- https://github.com/tttapa/docker-arm-cross-toolchain/releases/latest/download/x-tools-armv6-rpi-linux-
  gnueabihf.tar.xz | tar xJ -C ~/opt
```

```
$ mkdir -p ~/opt
$ wget -qO- https://github.com/tttapa/docker-arm-cross-toolchain/releases/latest/download/x-tools-armv8-rpi3-linux-
  gnueabihf.tar.xz | tar xJ -C ~/opt
```

```
$ mkdir -p ~/opt
$ wget -qO- https://github.com/tttapa/docker-arm-cross-toolchain/releases/latest/download/x-tools-aarch64-rpi3-linux-
  gnu.tar.xz | tar xJ -C ~/opt
```

For more details on how to select the correct one, and instructions for adding it to your PATH, see tttapa/docker-arm-cross-toolchain.

Building the toolchain yourself

If you want full control over the toolchain, customizing the compiler, debugger and other tools, the libc version, Linux version, etc., you can build the toolchain yourself, using crosstool-NG and the config files provided by tttapa/docker-arm-cross-toolchain.

Docker

As explained on the previous page, building the toolchain happens inside of a Docker container. This allows you to experiment in a sandbox-like environment. Starting from scratch is really easy, and you don't have to worry about messing up your main Linux installation. When you're done building, you can just delete the Docker containers and images.

Dockerfiles

A Dockerfile describes how the Docker image is built. In this project, we'll start from a standard Ubuntu image, install some build tools, and then compile the toolchain and the dependencies. Each step of the build process creates a new layer in the image. This is handy, because it means that if a build fails in one of the last steps, you can just fix it in your Dockerfile, and build it again. It'll then start from the last layer that was successfully built before, you don't have to start from the beginning (which would take a while, since we'll be building many large projects.)

The actual Dockerfiles used for the build can be found on GitHub at tttapa/docker-crosstool-ng-multiarch:Dockerfile and tttapa/docker-arm-cross-toolchain:Dockerfile, I'll briefly go over them on this page.

The following Dockerfile downloads, builds and installs crosstool-NG.

tttapa/docker-crosstool-ng-multiarch:Dockerfile

```
FROM ubuntu:bionic as ct-ng

# Install dependencies to build toolchain
RUN export DEBIAN_FRONTEND=noninteractive && \
    apt-get update && \
    apt-get install -y --no-install-recommends\
        gcc g++ gperf bison flex texinfo help2man make libncurses5-dev \
        python3-dev libtool automake libtool-bin gawk wget rsync git patch \
        unzip xz-utils bzip2 ca-certificates && \
    apt-get clean autoclean && \
    apt-get autoremove -y && \
    rm -rf /var/lib/apt/lists/*

# Add a user called `develop` and add him to the sudo group
RUN useradd -m develop && \
    echo "develop:develop" | chpasswd && \
    adduser develop sudo

USER develop
WORKDIR /home/develop

# Install autoconf
RUN wget https://ftp.gnu.org/gnu/autoconf/autoconf-2.71.tar.gz -O- | tar xz && \
    cd autoconf-2.71 && \
    ./configure --prefix=/home/develop/.local && \
    make -j$(nproc) && \
    make install && \
    cd .. && \
    rm -rf autoconf-2.71
ENV PATH=/home/develop/.local/bin:$PATH

# Download and install the latest version of crosstool-ng
RUN git clone -b master --single-branch --depth 1 \
        https://github.com/crosstool-ng/crosstool-ng.git
WORKDIR /home/develop/crosstool-ng
RUN git show --summary && \
    ./bootstrap && \
    mkdir build && cd build && \
    ../configure --prefix=/home/develop/.local && \
    make -j$(($(nproc) * 2)) && \
    make install &&  \
    cd .. && rm -rf build

ENV PATH=/home/develop/.local/bin:$PATH
WORKDIR /home/develop

# Patches
# https://www.raspberrypi.org/forums/viewtopic.php?f=91&t=280707&p=1700861#p1700861
RUN wget https://ftp.debian.org/debian/pool/main/b/binutils/binutils_2.40-2.debian.tar.xz -O- | \
    tar xJ debian/patches/129_multiarch_libpath.patch && \
    mkdir -p patches/binutils/2.40 && \
    mv debian/patches/129_multiarch_libpath.patch patches/binutils/2.40 && \
    rm -rf debian
```

You'll notice that the Dockerfile downloads a patch for the binutils package. This is done in order to support Debian Multiarch. Raspberry Pi OS and Ubuntu are both configured for Multiarch by default, so the toolchains we build must support this. To this end, we need to patch binutils. For more details, see this forum post.

Next, we build another Docker image, which actually builds the toolchain, using the crosstool-NG installation from the previous step.

tttapa/docker-arm-cross-toolchain:Dockerfile

```dockerfile
# Crosstool-NG ----------------------------------------------------------------

FROM --platform=$BUILDPLATFORM ubuntu:bionic AS ct-ng

# Install dependencies to build crosstool-ng and the toolchain
RUN export DEBIAN_FRONTEND=noninteractive && \
    apt-get update -y && \
    apt-get install -y --no-install-recommends \
        autoconf automake libtool-bin make texinfo help2man \
        sudo file gawk patch \
        python3 \
        g++ bison flex gperf \
        libncurses5-dev \
        perl libthread-queue-perl \
        ca-certificates wget git \
        bzip2 xz-utils unzip rsync && \
    apt-get clean autoclean && \
    apt-get autoremove -y && \
    rm -rf /var/lib/apt/lists/*

# Add a user called `develop` and add them to the sudo group
RUN useradd -m develop && echo "develop:develop" | chpasswd && \
    usermod -aG sudo develop

USER develop
WORKDIR /home/develop

# Install autoconf
RUN wget https://ftp.gnu.org/gnu/autoconf/autoconf-2.72.tar.gz -O- | tar xz && \
    cd autoconf-2.72 && \
    ./configure --prefix=/home/develop/.local && \
    make -j$(nproc) && \
    make install && \
    cd .. && \
    rm -rf autoconf-2.72
ENV PATH=/home/develop/.local/bin:${PATH}

# Build crosstool-ng
RUN git clone -b master --single-branch \
        https://github.com/crosstool-ng/crosstool-ng.git && \
    cd crosstool-ng && \
    git checkout f390dba6c73845389a3217169402d95a837fcee8 && \
    git show --summary && \
    ./bootstrap && \
    mkdir build && cd build && \
    ../configure --prefix=/home/develop/.local && \
    make -j$(($(nproc) * 2)) && \
    make install &&  \
    cd /home/develop && rm -rf crosstool-ng

# Patches
COPY --chown=develop:develop patches patches
# https://www.raspberrypi.org/forums/viewtopic.php?f=91&t=280707&p=1700861#p1700861
RUN wget https://ftp.debian.org/debian/pool/main/b/binutils/binutils_2.45-3.debian.tar.xz -O- | \
    tar xJ debian/patches/129_multiarch_libpath.patch && \
    mkdir -p patches/binutils/2.45 && \
    mv debian/patches/129_multiarch_libpath.patch patches/binutils/2.45 && \
    rm -rf debian

# Toolchain -------------------------------------------------------------------

FROM --platform=$BUILDPLATFORM ct-ng AS gcc-build

ARG HOST_TRIPLE
ARG GCC_VERSION
ARG PKG_VERSION

# Build the toolchain
COPY --chown=develop:develop ${HOST_TRIPLE}.defconfig .
COPY --chown=develop:develop ${HOST_TRIPLE}.env .
RUN [ -n "${GCC_VERSION}" ] && { echo "CT_GCC_V_${GCC_VERSION}=y" >> ${HOST_TRIPLE}.defconfig; }
RUN [ -n "${PKG_VERSION}" ] && { echo "CT_TOOLCHAIN_PKGVERSION=\"tttapa/docker-arm-cross-
toolchain:${HOST_TRIPLE}@${PKG_VERSION}\"" >> ${HOST_TRIPLE}.defconfig; }
RUN cp ${HOST_TRIPLE}.defconfig defconfig && ct-ng defconfig
RUN . ./${HOST_TRIPLE}.env && \
    ct-ng build || { cat build.log && false; } && rm -rf .build

RUN chmod +w /home/develop/x-tools/${HOST_TRIPLE}
COPY --chown=develop:develop cmake/Common.toolchain.cmake /home/develop/x-tools/${HOST_TRIPLE}/
COPY --chown=develop:develop cmake/${HOST_TRIPLE}/* /home/develop/x-tools/${HOST_TRIPLE}/
RUN chmod -w /home/develop/x-tools/${HOST_TRIPLE}

# Toolchain (Canadian) --------------------------------------------------------

FROM --platform=$BUILDPLATFORM gcc-build AS gcc-build-canadian

ARG HOST_TRIPLE
ARG TARGET_TRIPLE=${HOST_TRIPLE}
ARG GCC_VERSION

# Add cross toolchain for host to PATH
RUN mkdir -p opt && chmod -R +w /home/develop/x-tools && mv /home/develop/x-tools /home/develop/opt
```

```
ENV HOST_TOOLCHAIN_PATH=/home/develop/opt/x-tools/${HOST_TRIPLE}
ENV PATH=${HOST_TOOLCHAIN_PATH}/bin:${PATH}

# Build the toolchain
COPY --chown=develop:develop ${TARGET_TRIPLE}.defconfig .
COPY --chown=develop:develop ${TARGET_TRIPLE}.env .
RUN [ -n "${GCC_VERSION}" ] && { echo "CT_GCC_V_${GCC_VERSION}=y" >> ${TARGET_TRIPLE}.defconfig; }
RUN [ -n "${PKG_VERSION}" ] && { echo "CT_TOOLCHAIN_PKGVERSION=\"tttapa/docker-arm-cross-toolchain:${HOST_TRIPLE}@${PKG_VERSION}\"" >> ${TARGET_TRIPLE}.defconfig; }
RUN echo "CT_CANADIAN=y" >> ${TARGET_TRIPLE}.defconfig && \
    echo "CT_HOST=\"${HOST_TRIPLE}\"" >> ${TARGET_TRIPLE}.defconfig
RUN cat ${TARGET_TRIPLE}.defconfig && \
    cp ${TARGET_TRIPLE}.defconfig defconfig && ct-ng defconfig
RUN . ./${TARGET_TRIPLE}.env && \
    ct-ng build || { cat build.log && false; } && rm -rf .build

RUN chmod +w /home/develop/x-tools/HOST-${HOST_TRIPLE}/${TARGET_TRIPLE}
COPY --chown=develop:develop cmake/Common.toolchain.cmake /home/develop/x-tools/HOST-${HOST_TRIPLE}/${TARGET_TRIPLE}/
COPY --chown=develop:develop cmake/${TARGET_TRIPLE}/* /home/develop/x-tools/HOST-${HOST_TRIPLE}/${TARGET_TRIPLE}/
RUN chmod -w /home/develop/x-tools/HOST-${HOST_TRIPLE}/${TARGET_TRIPLE}

# Build container (base) ------------------------------------------------------

FROM ubuntu:noble AS gcc-dev-base

RUN export DEBIAN_FRONTEND=noninteractive && \
    apt-get update -y && \
    apt-get install --no-install-recommends -y \
        ninja-build cmake make bison flex \
        tar xz-utils gzip zip unzip bzip2 zstd \
        ca-certificates wget git sudo file && \
    apt-get clean autoclean && \
    apt-get autoremove -y && \
    rm -rf /var/lib/apt/lists/*

# Add a user called `develop` and add them to the sudo group
RUN useradd -m develop && echo "develop:develop" | chpasswd && \
    usermod -aG sudo develop

USER develop
WORKDIR /home/develop

# Build container (cross) -----------------------------------------------------

FROM gcc-dev-base AS gcc-dev-cross

ARG HOST_TRIPLE
ARG TARGET_TRIPLE=${HOST_TRIPLE}

ENV TOOLCHAIN_PATH=/home/develop/opt/x-tools/${TARGET_TRIPLE}
ENV PATH=${TOOLCHAIN_PATH}/bin:${PATH}

# Copy the toolchain
COPY --chown=develop:develop --from=gcc-build-canadian /home/develop/x-tools/HOST-${HOST_TRIPLE}/${TARGET_TRIPLE} ${TOOLCHAIN_PATH}
RUN ${TARGET_TRIPLE}-g++ --version

# Build container -------------------------------------------------------------

FROM gcc-dev-base AS gcc-dev

ARG HOST_TRIPLE

ENV TOOLCHAIN_PATH=/home/develop/opt/x-tools/${HOST_TRIPLE}
ENV PATH=${TOOLCHAIN_PATH}/bin:${PATH}

# Copy the toolchain
COPY --chown=develop:develop --from=gcc-build /home/develop/x-tools/${HOST_TRIPLE} ${TOOLCHAIN_PATH}
RUN ${HOST_TRIPLE}-g++ --version
```

Different variants of the Raspberry Pi use different configuration files. These files have names that contain the full target triplet. They can be found on GitHub in same folder as the Dockerfile.

Building the crosstool-NG base image

This step is optional. It can be useful to build it yourself if you want the very latest version of crosstool-NG.

```
$ cd docker-crosstool-ng-multiarch
$ docker build . -t ghcr.io/tttapa/docker-crosstool-ng-multiarch:master
```

If you skip this step, pull the image from GitHub:

```
$ docker pull ghcr.io/tttapa/docker-crosstool-ng-multiarch:master
```

Upgrading and customizing the configurations

Next, we'll use crosstool-NG to update the configuration files and interactively customize the configuration. Here you can choose versions of the compiler and other tools, Linux and libc versions, and so on.

First, start a shell in the crosstool-NG container you built or pulled in the previous step. Mount the `docker-arm-cross-toolchain` folder containing the config files, so you can access them inside of the container.

```
$  cd ../docker-arm-cross-toolchain
$  docker run -it -v"$PWD:/mnt" ghcr.io/tttapa/docker-crosstool-ng-multiarch:master
```

Now copy the configuration you need to a file named `.config`, update the configuration, make your changes, save the changes, and then overwrite the previous configuration with the new one. Here I'll use the `aarch64-rpi3-linux-gnu` configuration, but you should use the correct one for your specific setup.

```
[docker] $  # Rename the old configuration
[docker] $  cp /mnt/aarch64-rpi3-linux-gnu.config .config
[docker] $  # Upgrade the configuration
[docker] $  ct-ng upgradeconfig
[docker] $  # Customize the configuration
[docker] $  ct-ng menuconfig
[docker] $  # Overwrite the old configuration
[docker] $  cp .config /mnt/aarch64-rpi3-linux-gnu.config
[docker] $  exit
```

Building the toolchain

Finally, build the toolchain with the new config. Use the host triple that matches the configuration file name.

```
$  docker build . --build-arg=HOST_TRIPLE=aarch64-rpi3-linux-gnu -t ghcr.io/tttapa/docker-arm-cross-toolchain:aarch64-rpi3-
   linux-gnu
```

Packaging and extracting the toolchain

Finally, create a tar archive of the toolchain, and copy it out of the Docker container so you can use it.

```
$  container=$(docker run -d ghcr.io/tttapa/docker-arm-cross-toolchain:aarch64-rpi3-linux-gnu bash -c "tar cJf x-tools.tar.xz
   x-tools")
$  docker wait $container
$  docker cp $container:/home/develop/x-tools.tar.xz x-tools-aarch64-rpi3-linux-gnu.tar.xz
$  docker rm $container
```

Installing the toolchain

You can now simply extract the toolchain and install it somewhere on your system, e.g. in `~/opt`:

```
$  mkdir -p ~/opt
$  tar xJf x-tools-aarch64-rpi3-linux-gnu.tar.xz -C ~/opt
```

For more details on how to select the correct one, and instructions for adding it to your PATH, see tttapa/docker-arm-cross-toolchain.