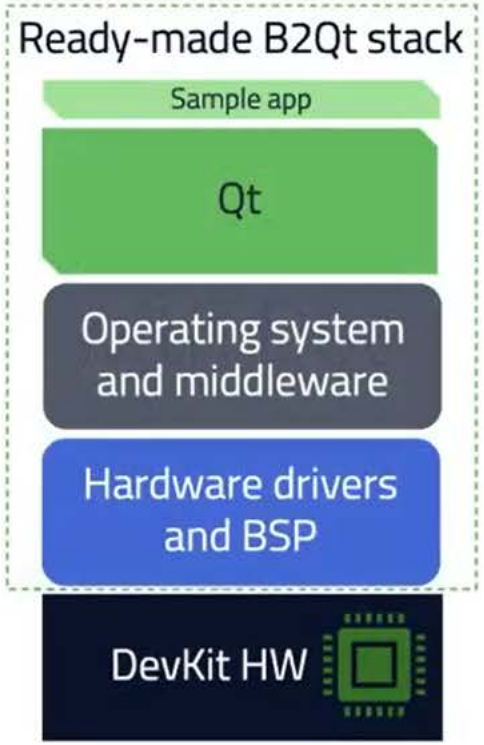
Build and Run Embedded Apps Faster from QtCreator with Docker

July 8,2020

Building Qt SW for Embedded Targets

- Based on Yocto Embedded Linux recipes
- Created embedded toolchains can be configured as Qt Creator kits
- Qt Debug Bridge helps makes debugging and deployment rather straightforward
- Boot 2 Qt - Plenty of ready-made images for variety of development boards
- How to improve the workflow with containers?

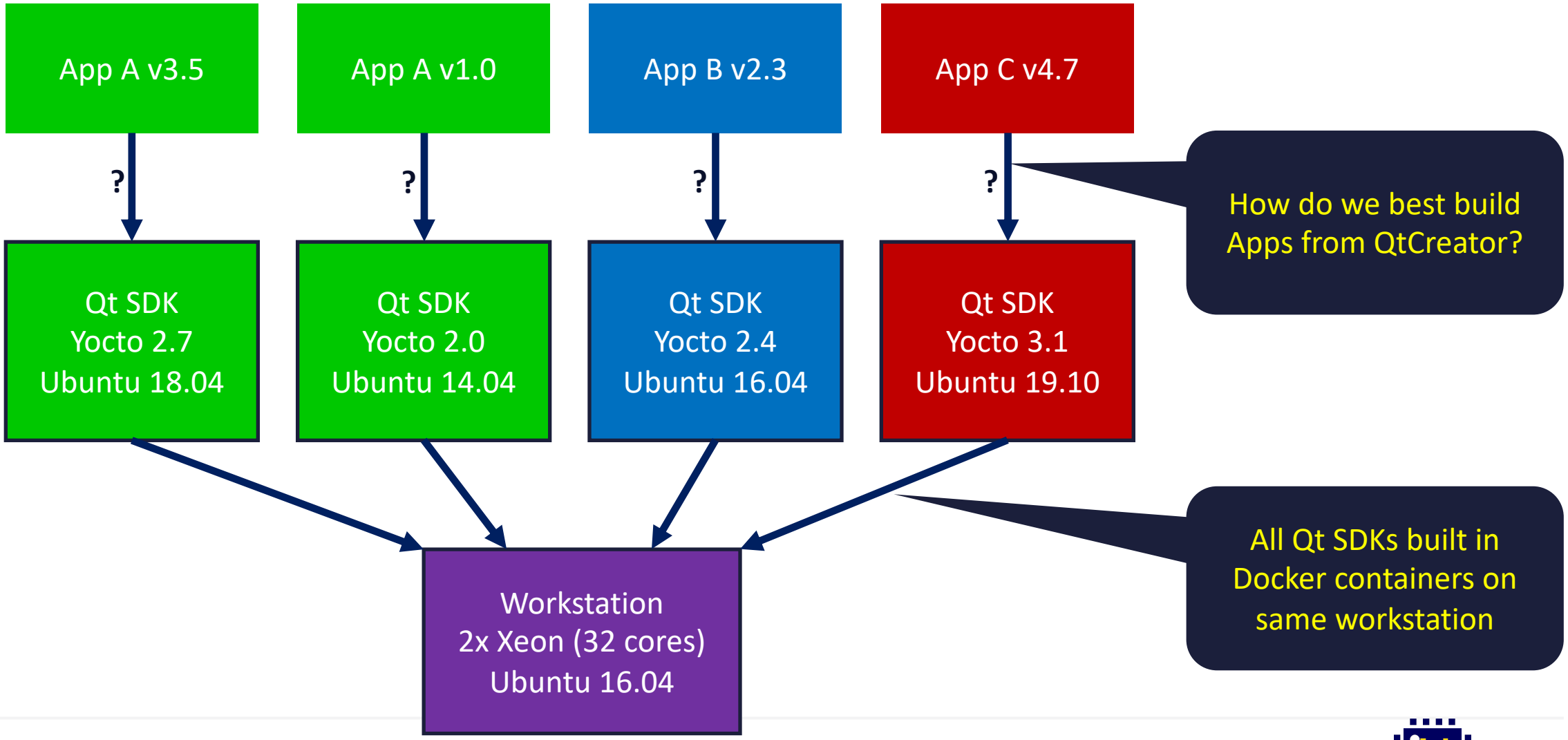


Build and Run Embedded Apps Faster from QtCreator with Docker

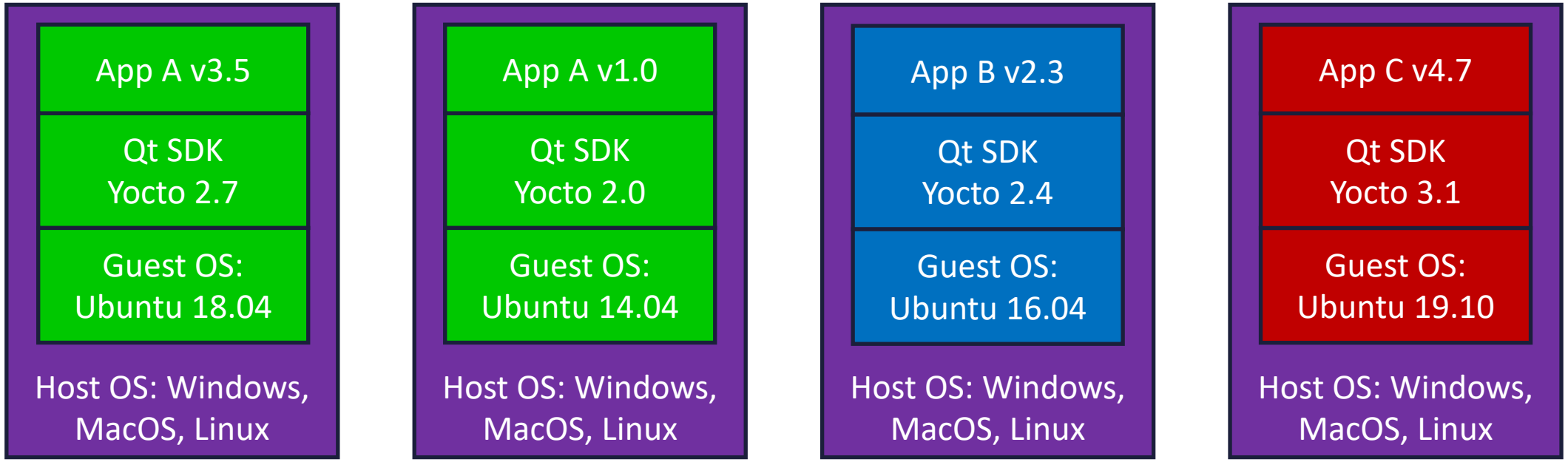
Burkhard Stubert

As a solo consultant, I help teams succeed with Qt embedded systems

The Problem


App A v3.5
App A v1.0
App B v2.3
App C v4.7
?
?
?
?
Qt SDK
Yocto 2.7
Ubuntu 18.04
Qt SDK
Yocto 2.0
Ubuntu 14.04
Qt SDK
Yocto 2.4
Ubuntu 16.04
Qt SDK
Yocto 3.1
Ubuntu 19.10
How do we best build Apps from QtCreator?
Workstation
2x Xeon (32 cores)
Ubuntu 16.04
All Qt SDKs built in Docker containers on same workstation

Solution 1 (Currently Used): Virtual Machines

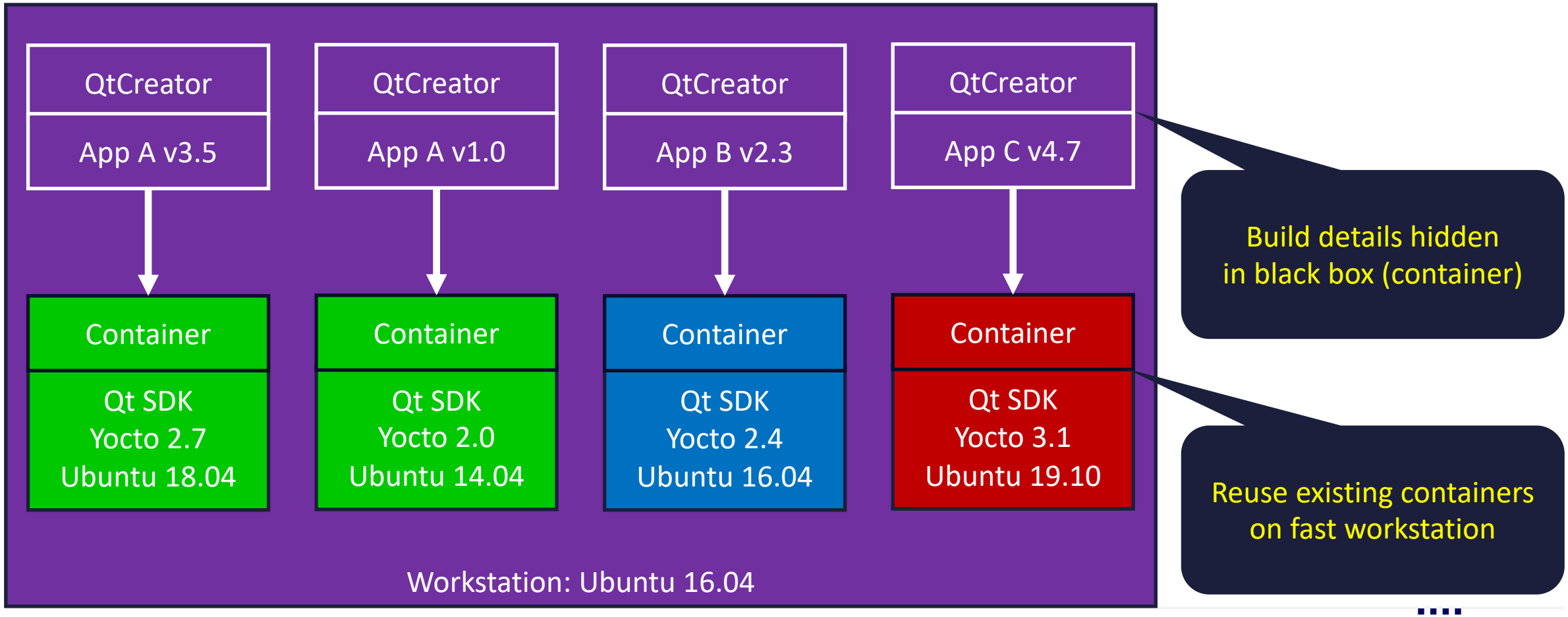


2-3 VMs fit on same dev PC

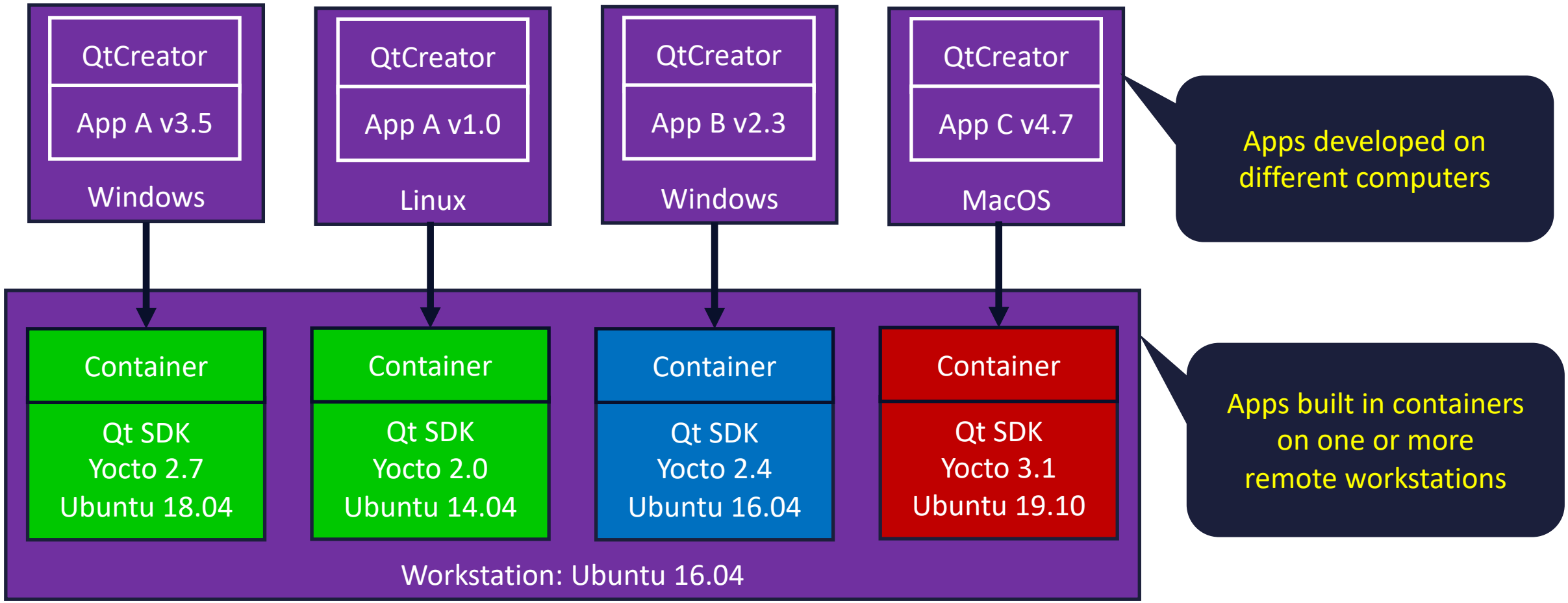
VMs slower than containers

Fast workstation and existing containers not used for app builds

Solution 2 (This Talk): Containers on Local Workstation



Solution 3 (Future): Containers on Remote Workstation



QtCreator-CMake-Docker

- Motivation
- Idea: Docker Wrapper for CMake
- Prerequisites
  - General
  - SSH Access to Device
- Building App with Docker
  - Installing Qt SDK in Container
  - Configuring QtCreator
  - Building App with Docker-CMake
- Running App on Device
  - Deployment and Run Settings
  - Running App

How QtCreator Calls CMake (Native Build)

Stage	Working Directory	CMake Command
Configure project	**/tmp**/QtCreator-Yqhjyl/qtc-cmake-MDHAJjOH => <work-dir-1>	cmake '-GUnix Makefiles' -C <work-dir-1>/qtcsettings.cmake /public/Work/cuteradio-apps
Generate build files	**/public/Work**/build-cuteradio-apps-Desktop_Qt_5_14_2_GCC_64bit-Debug => <work-dir-2>	cmake '-GUnix Makefiles' -C <work-dir-2>/qtcsettings.cmake /public/Work/cuteradio-app
Compile	<work-dir-2>	cmake --build . --target all -- -j4
Install	<work-dir-2>	cmake --build . --target install

Docker Wrapper for CMake

```
#!/bin/bash

args=$(echo $@ | sed -e "s|-GCodeBlocks - Unix Makefiles|'-
GCodeBlocks - Unix Makefiles'|g")

docker run --rm -v /public/Work:/public/Work -v /tmp:/tmp \
    -w $(pwd) dr-yocto:sdk-18.04 cmake $args
```

Resurrect single quotes removed by shell

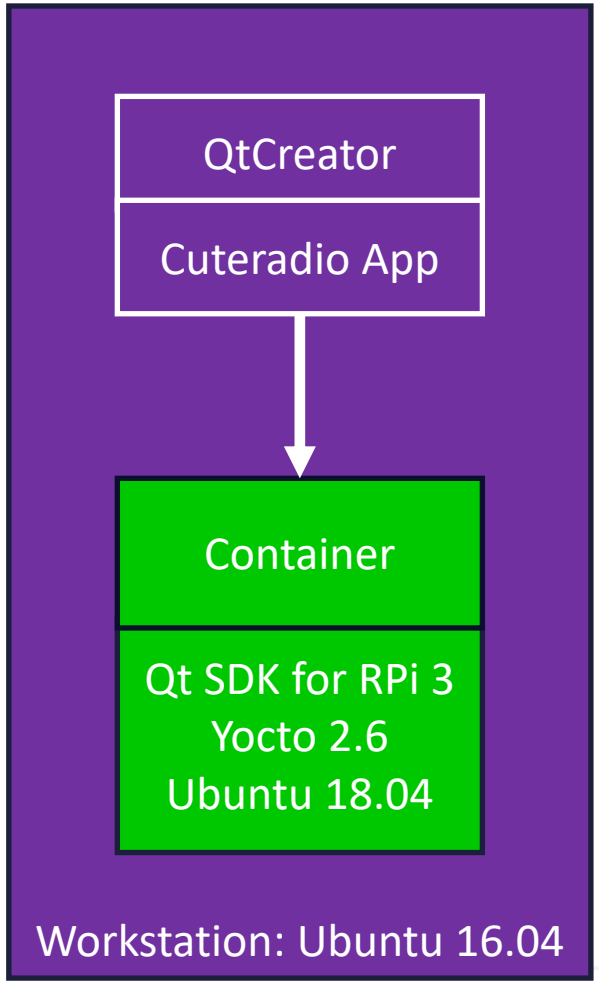
Save script in $HOME/bin/dr-cmake

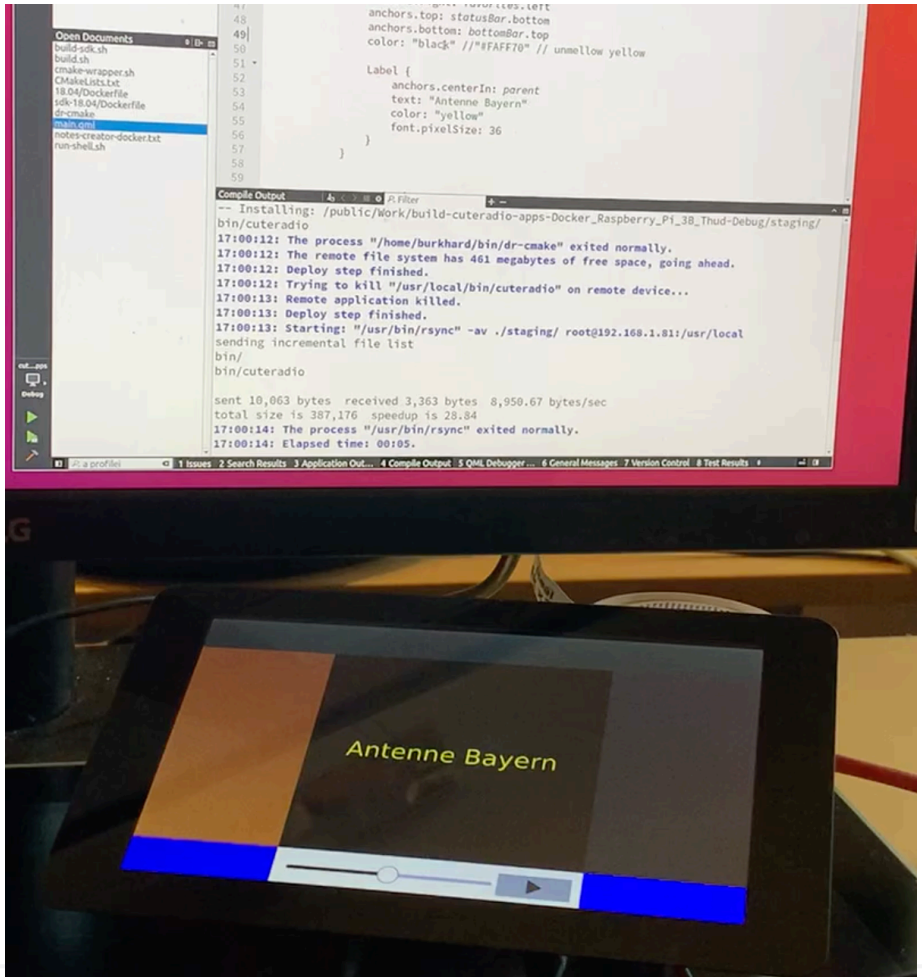
/public/Work and /tmp visible both on host PC and in container

QtCreator-CMake-Docker

- Motivation
- Idea: Docker Wrapper for CMake
- Prerequisites
  - General
  - SSH Access to Device
- Building App with Docker
  - Installing Qt SDK in Container
  - Configuring QtCreator
  - Building App with Docker-CMake
- Running App on Device
  - Deployment and Run Settings
  - Running App

My Setup





Only a proof of concept!

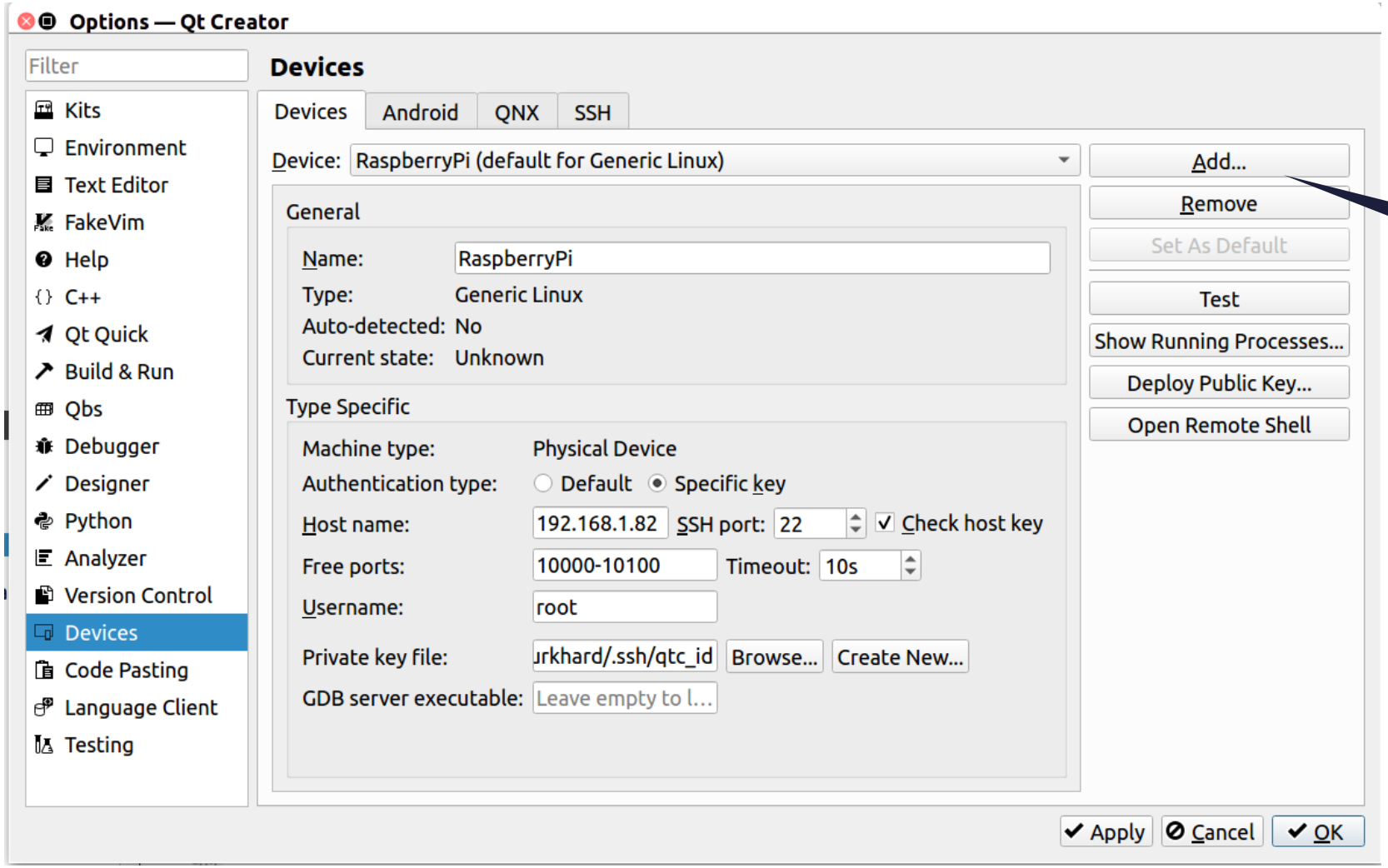
Prerequisites: General

- Install Docker on workstation (see [2])
- Create a Docker container (sdk-18.04) for Yocto builds (see [3])
- Build a Linux image with the Docker container (see [3])
  - Or: Use a pre-built Boot2Qt image from The Qt Company (see [5] and [6])
- Build Qt SDK with the Docker container (see [4])
  - Or: Use a pre-built Boot2Qt SDK from The Qt Company (see [5] and [6])
- Establish SSH connection between workstation and device (see [1] and [4])

QtCreator-CMake-Docker

- Motivation
- Idea: Docker Wrapper for CMake
- Prerequisites
  - General
  - SSH Access to Device
- Building App with Docker
  - Installing Qt SDK in Container
  - Configuring QtCreator
  - Building App with Docker-CMake
- Running App on Device
  - Deployment and Run Settings
  - Running App

Prerequisites: SSH Access to Device



See [1] and [4]
how to add device

QtCreator-CMake-Docker

- Motivation
- Idea: Docker Wrapper for CMake
- Prerequisites
  - General
  - SSH Access to Device
- Building App with Docker
  - Installing Qt SDK in Container
  - Configuring QtCreator
  - Building App with Docker-CMake
- Running App on Device
  - Deployment and Run Settings
  - Running App

Installing Qt SDK in Container

On Workstation:

```
$ cd /public/Work
$ docker run –t –rm –v /public/Work:/public/Work -v /tmp:/tmp \
    -w $(pwd) dr-yocto:sdk-18.04
```

In Docker container:

```
# cd cuteradio-thud/build-rpi3/tmp/deploy/sdk/
# ./poky-glibc-x86_64-meta-toolchain-qt5-cortexa7t2hf-neon-vfpv4-
toolchain-2.6.4.sh
Poky (Yocto Project Reference Distro) SDK installer version 2.6.4
==========================================================
Enter target directory for SDK (default: /opt/poky/2.6.4):
/public/Work/qt-sdk-thud
...
```

Setting Up Build Environment in Dockerfile

source environment-setup-cortexa7t2hf-neon-vfpv4-poky-linux-gnueabi

```
export OECORE_NATIVE_SYSROOT="/public/Work/qt-sdk-thud/sysroots/x86_64-pokysdk-linux"
export OECORE_TARGET_SYSROOT="$SDKTARGETSYSROOT"
export OECORE_BASELIB="lib"
export OECORE_TARGET_ARCH="arm"
export OECORE_TARGET_OS="linux-gnueabi"
...
```

Change all environment variables

Dockerfile

```
ENV OECORE_NATIVE_SYSROOT="/public/Work/qt-sdk-thud/sysroots/x86_64-pokysdk-linux"
ENV OECORE_TARGET_SYSROOT="${SDKTARGETSYSROOT}"
ENV OECORE_BASELIB="lib"
ENV OECORE_TARGET_ARCH="arm"
ENV OECORE_TARGET_OS="linux-gnueabi"
...
```

Fixing Environment Variables for QtCreator

```
export CC="arm-poky-linux-gnueabi-gcc  -march=armv7ve -mthumb -mfpu=neon-vfpv4 -
mfloat-abi=hard -mcpu=cortex-a7 --sysroot=$SDKTARGETSYSROOT"
export CFLAGS=" -O2 -pipe -g -feliminate-unused-debug-types "

export CXX="arm-poky-linux-gnueabi-g++  -march=armv7ve -mthumb -mfpu=neon-vfpv4 -
mfloat-abi=hard -mcpu=cortex-a7 --sysroot=$SDKTARGETSYSROOT"
export CXXFLAGS=" -O2 -pipe -g -feliminate-unused-debug-types "
```

Move options from CC to CFLAGS and from CXX to CXXFLAGS

Do same with LD/LDFLAGS and CPP/CPPFLAGS

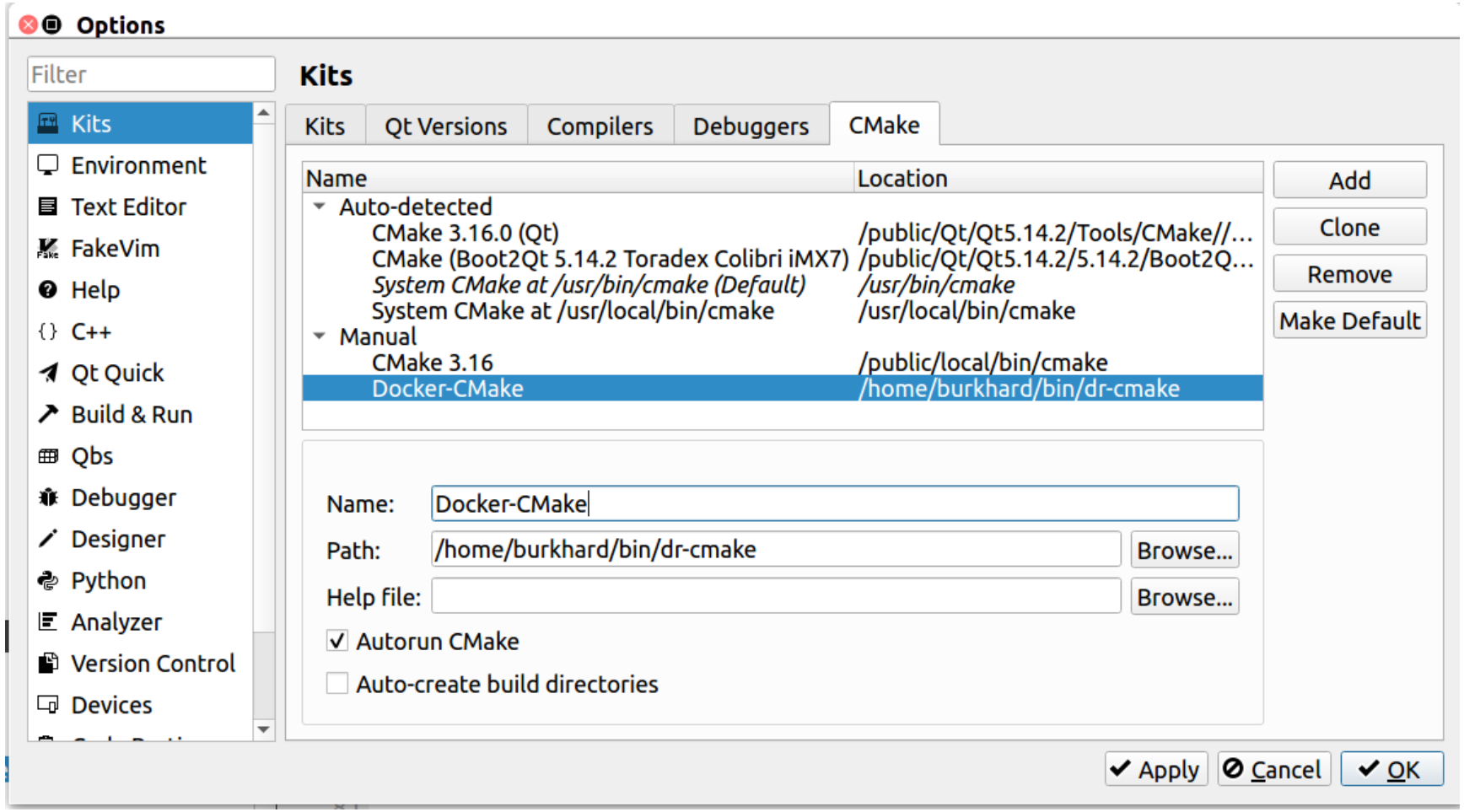
```
ENV CC="arm-poky-linux-gnueabi-gcc"
ENV CFLAGS=" -O2 -pipe -g -feliminate-unused-debug-types  -march=armv7ve -mthumb -
mfpu=neon-vfpv4 -mfloat-abi=hard -mcpu=cortex-a7 --sysroot=${SDKTARGETSYSROOT}"

ENV CXX="arm-poky-linux-gnueabi-g++"
ENV CXXFLAGS=" -O2 -pipe -g -feliminate-unused-debug-types -march=armv7ve -mthumb -
mfpu=neon-vfpv4 -mfloat-abi=hard -mcpu=cortex-a7 --sysroot=${SDKTARGETSYSROOT}"
```

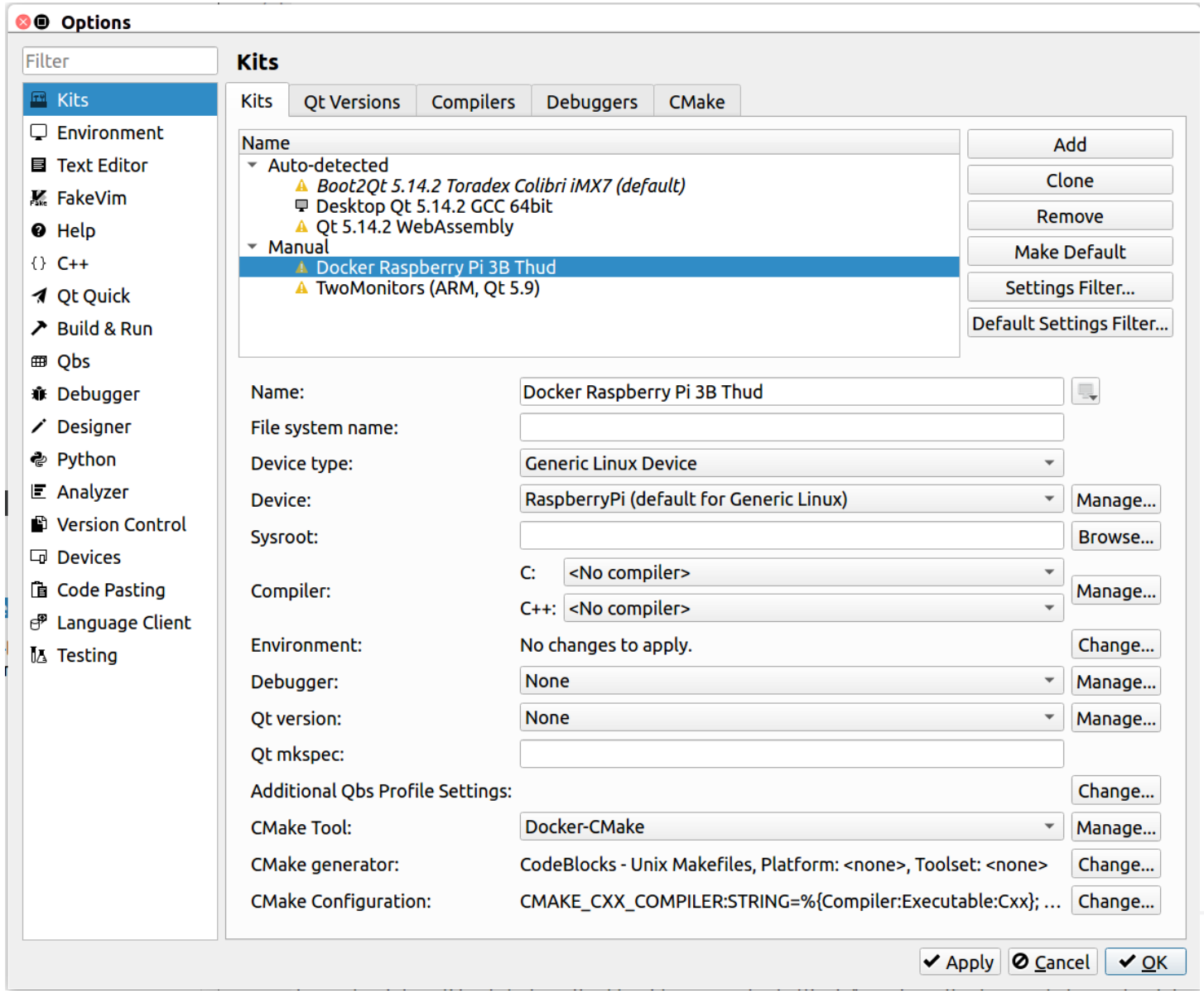
QtCreator-CMake-Docker

- Motivation
- Idea: Docker Wrapper for CMake
- Prerequisites
  - General
  - SSH Access to Device
- Building App with Docker
  - Installing Qt SDK in Container
  - Configuring QtCreator
  - Building App with Docker-CMake
- Running App on Device
  - Deployment and Run Settings
  - Running App

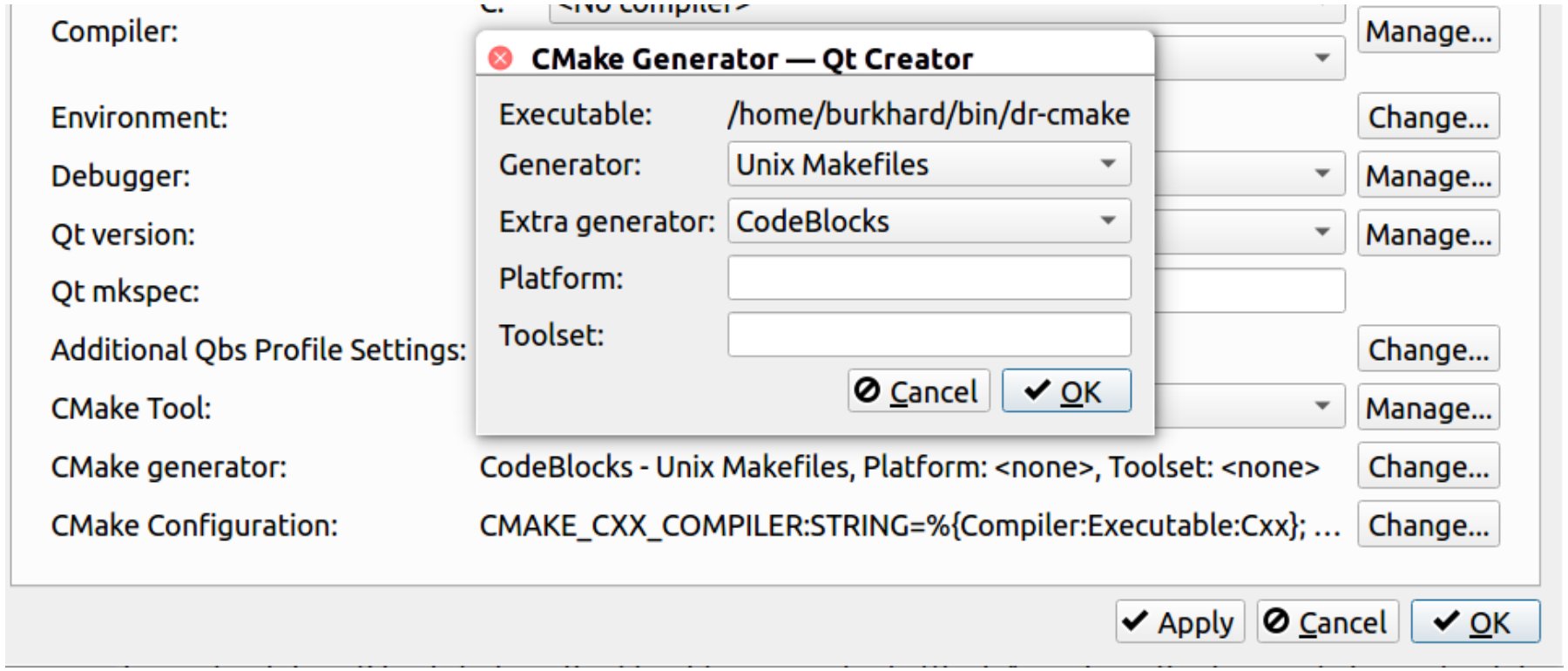
Configuring QtCreator: CMake



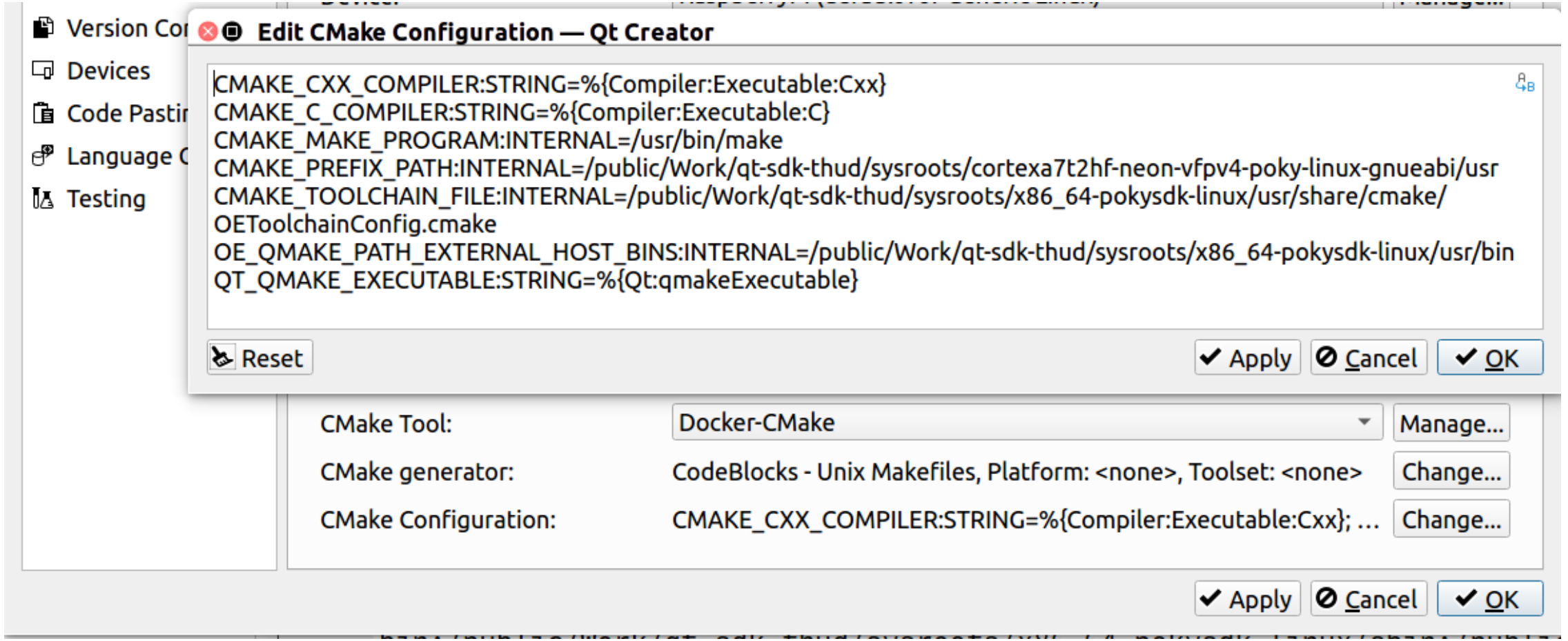
Configuring QtCreator: Kit



Configuring QtCreator: Kit – CMake Generator



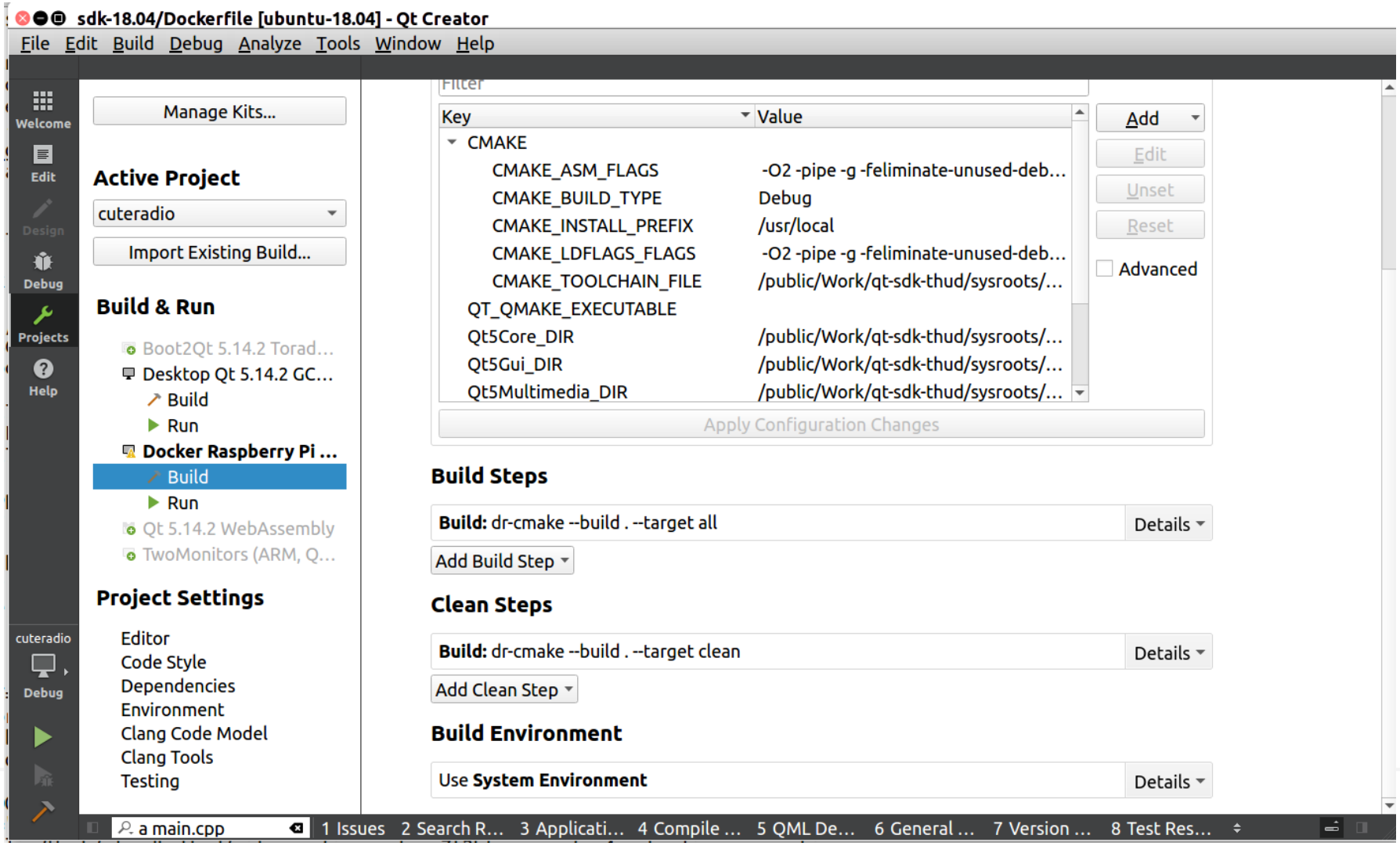
Configuring QtCreator:
Kit – CMake Configuration



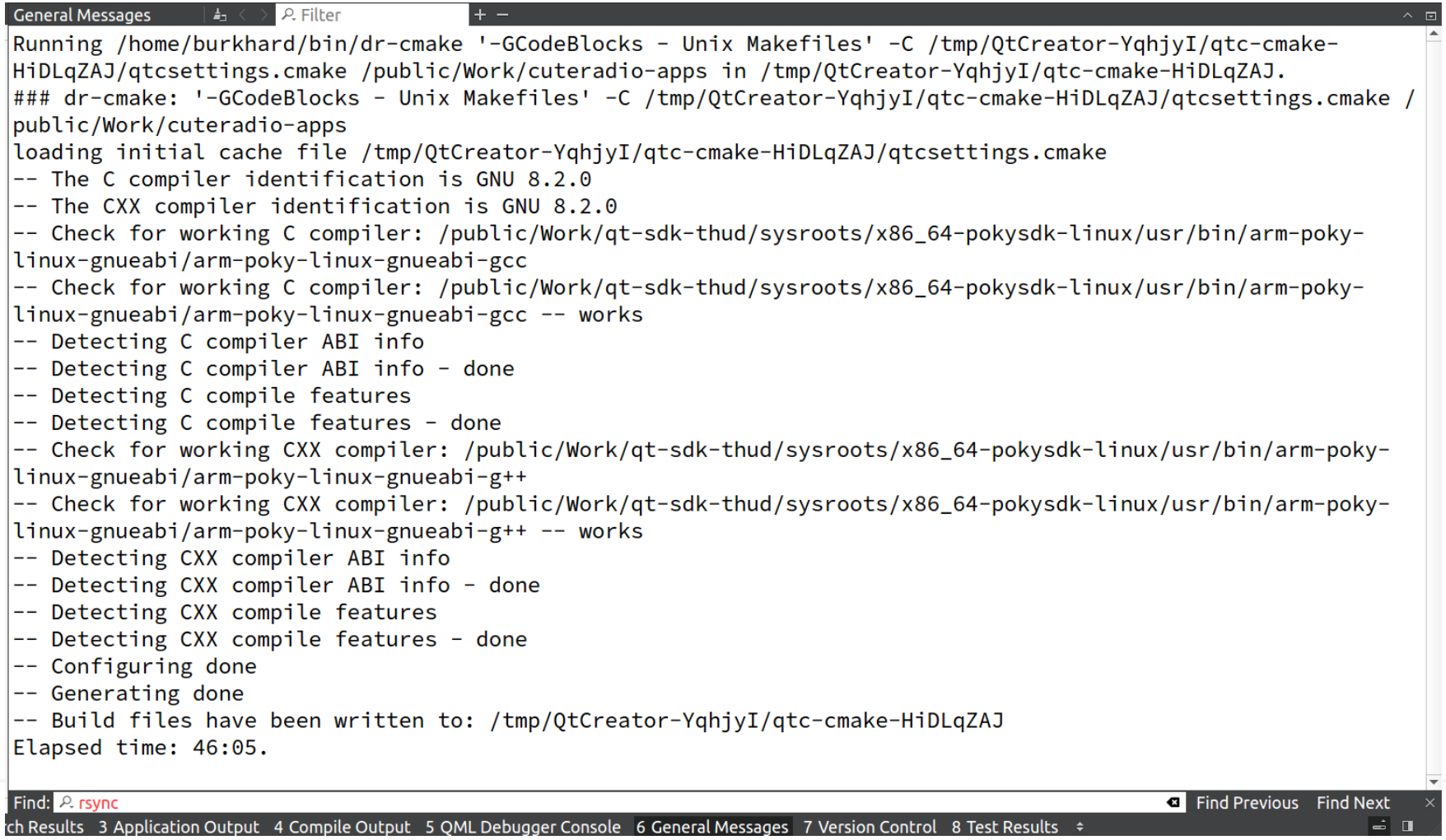
QtCreator-CMake-Docker

- Motivation
- Idea: Docker Wrapper for CMake
- Prerequisites
  - General
  - SSH Access to Device
- Building App with Docker
  - Installing Qt SDK in Container
  - Configuring QtCreator
  - Building App with Docker-CMake
- Running App on Device
  - Deployment and Run Settings
  - Running App

Building the App with dr-cmake: Switching to Project "Docker Raspberry Pi"



Building the App with dr-cmake: Output when Switching the Project



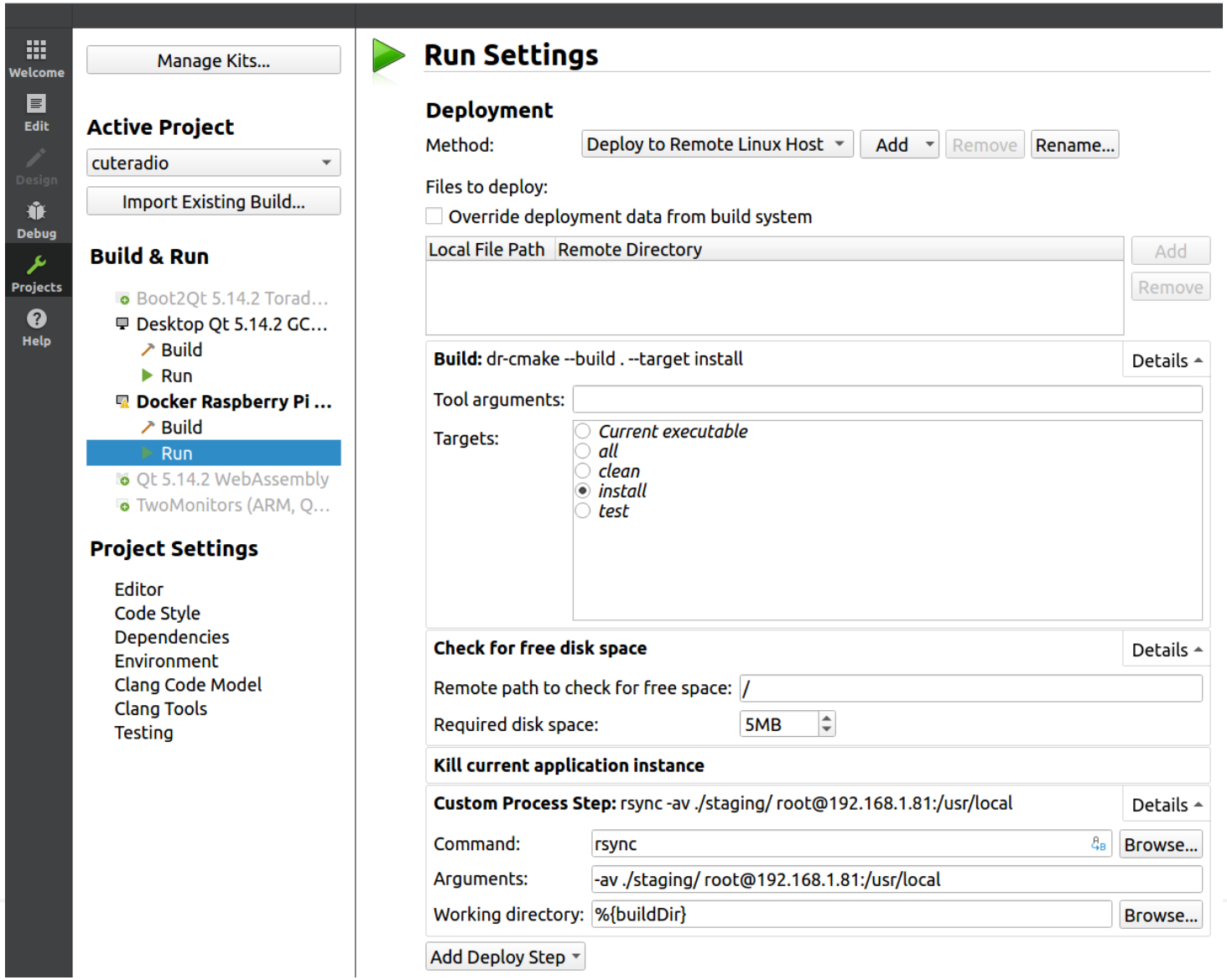
Building the App with dr-cmake: CMake-Output of Switching to Project

```
15:50:29: Running steps for project cuteradio-apps...
15:50:30: Starting: "/home/burkhard/bin/dr-cmake" --build . --target all
[ 16%] Automatic MOC for target cuteradio
[ 16%] Built target cuteradio_autogen
[ 33%] Automatic RCC for qml.qrc
Scanning dependencies of target cuteradio
[ 50%] Building CXX object
CMakeFiles/cuteradio.dir/cuteradio_autogen/mocs_compilation.cpp.o
[ 66%] Building CXX object CMakeFiles/cuteradio.dir/main.cpp.o
[ 83%] Building CXX object
CMakeFiles/cuteradio.dir/cuteradio_autogen/EWIEGA46WW/qrc_qml.cpp.o
[100%] Linking CXX executable cuteradio
[100%] Built target cuteradio
15:50:33: The process "/home/burkhard/bin/dr-cmake" exited normally.
15:50:33: Elapsed time: 00:05.
```

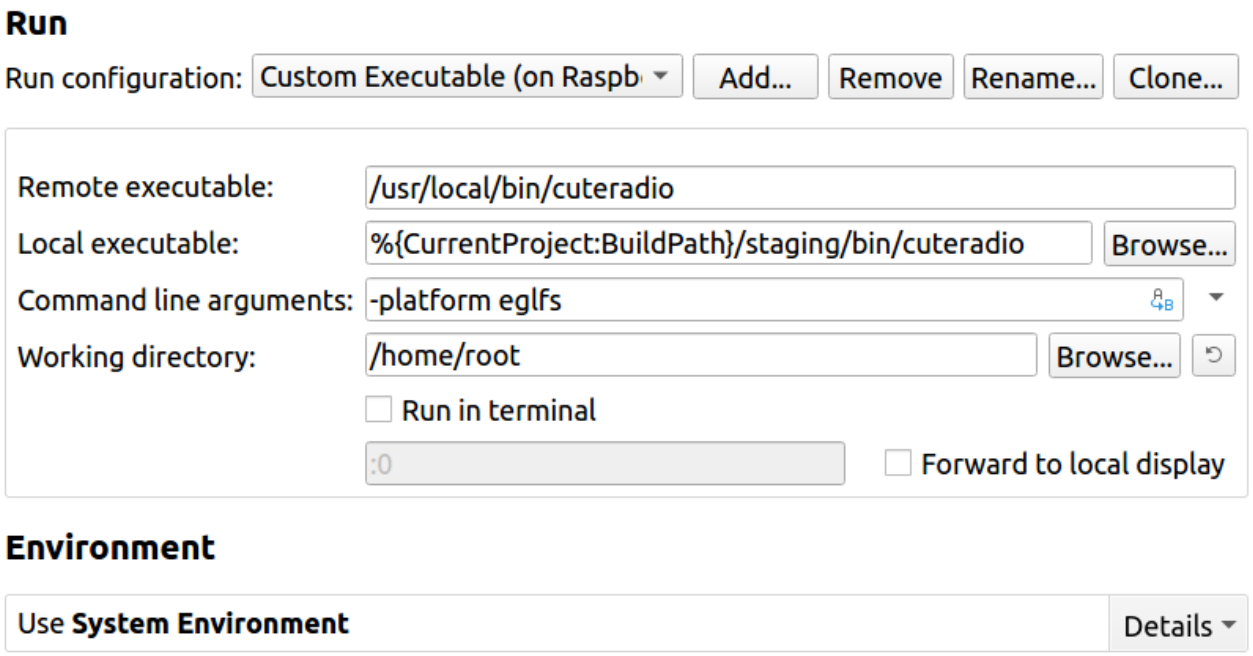
QtCreator-CMake-Docker

- Motivation
- Idea: Docker Wrapper for CMake
- Prerequisites
  - General
  - SSH Access to Device
- Building App with Docker
  - Installing Qt SDK in Container
  - Configuring QtCreator
  - Building App with Docker-CMake
- Running App on Device
  - Deployment and Run Settings
  - Running App

Deployment Settings



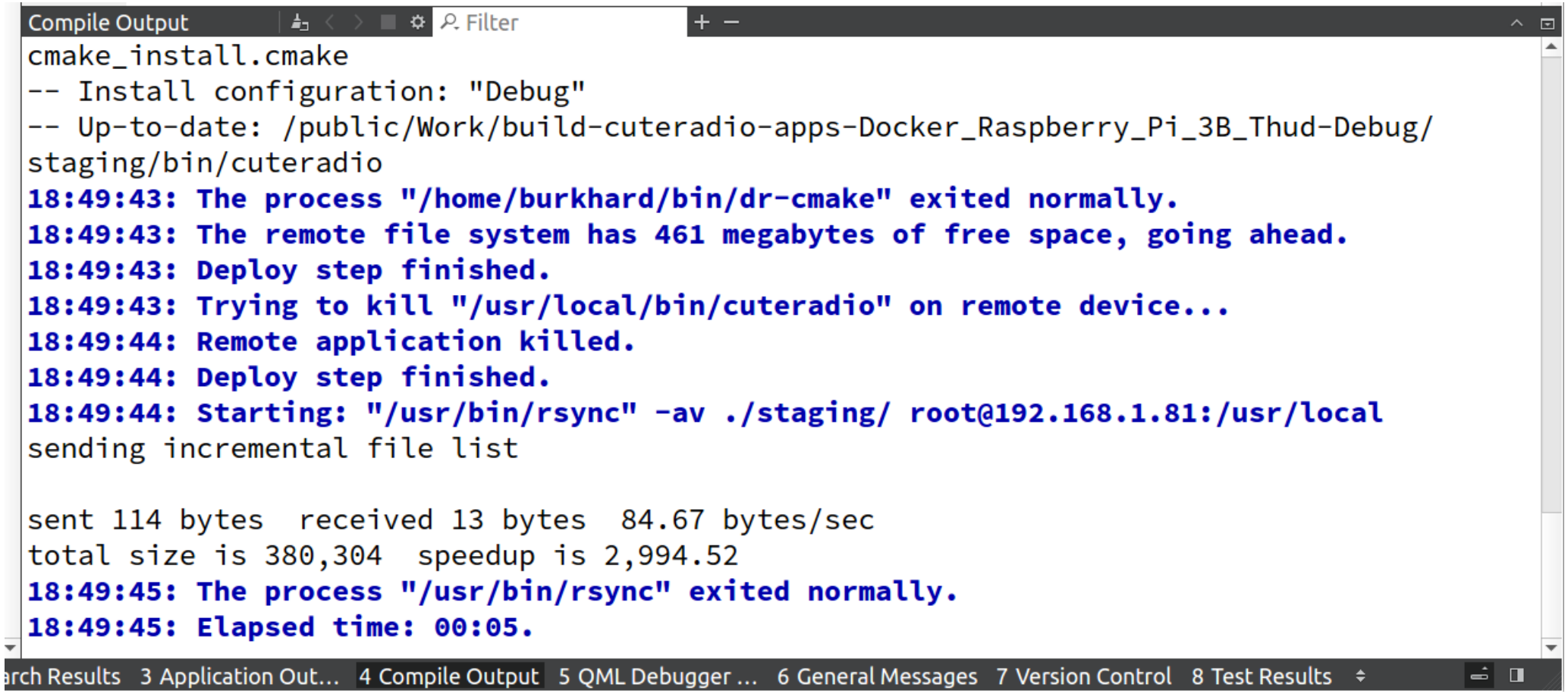
Run Settings



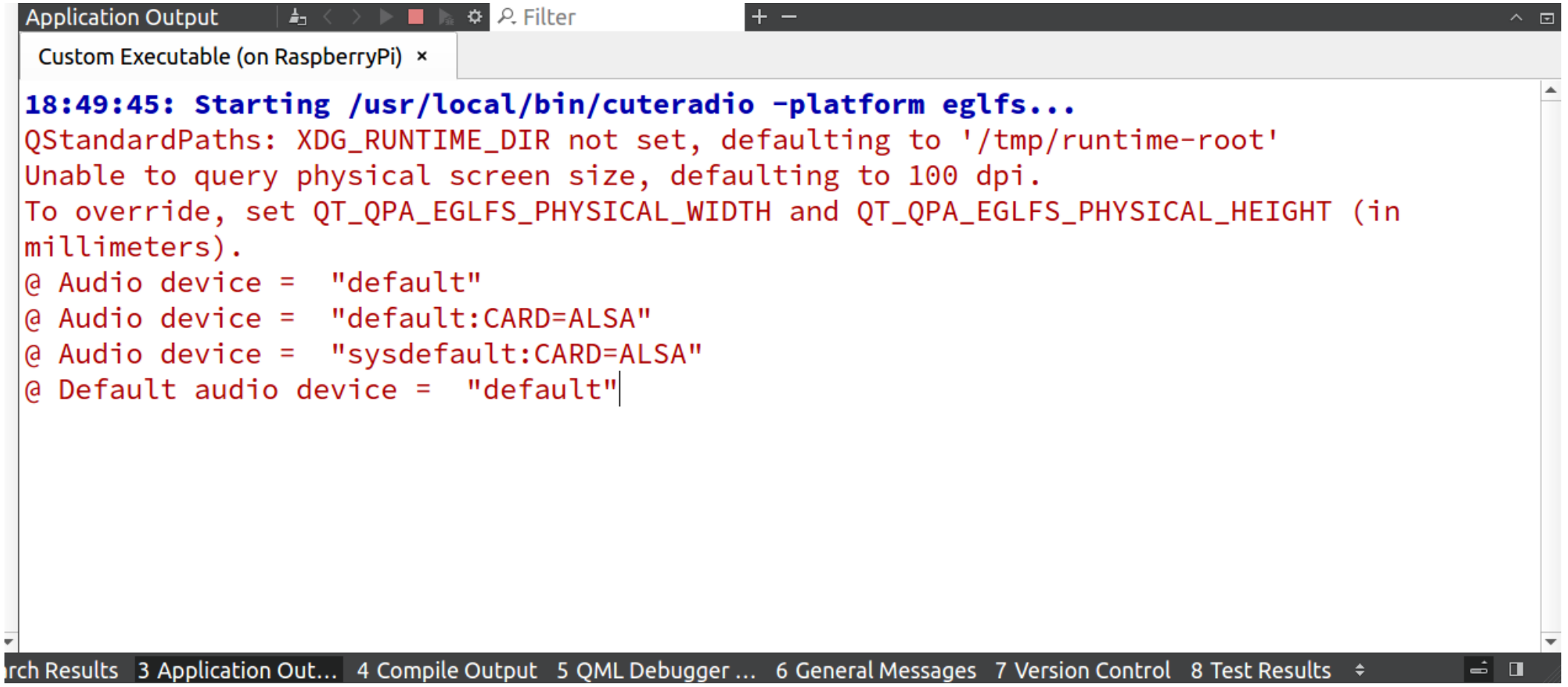
QtCreator-CMake-Docker

- Motivation
- Idea: Docker Wrapper for CMake
- Prerequisites
  - General
  - SSH Access to Device
- Building App with Docker
  - Installing Qt SDK in Container
  - Configuring QtCreator
  - Building App with Docker-CMake
- Running App on Device
  - Deployment and Run Settings
  - Running App

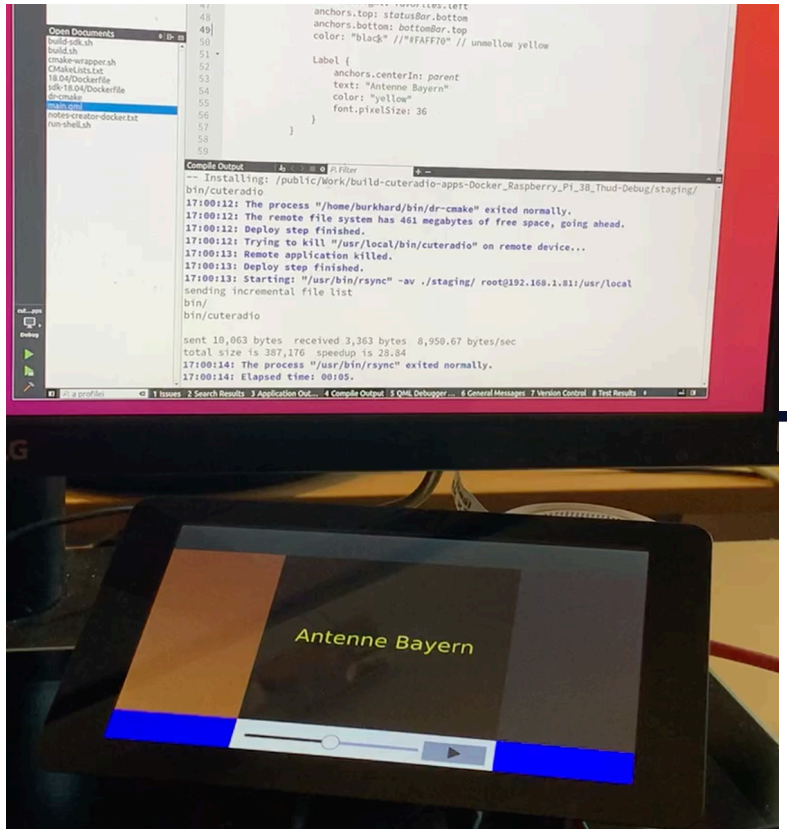
Running App on Device: Compile Output



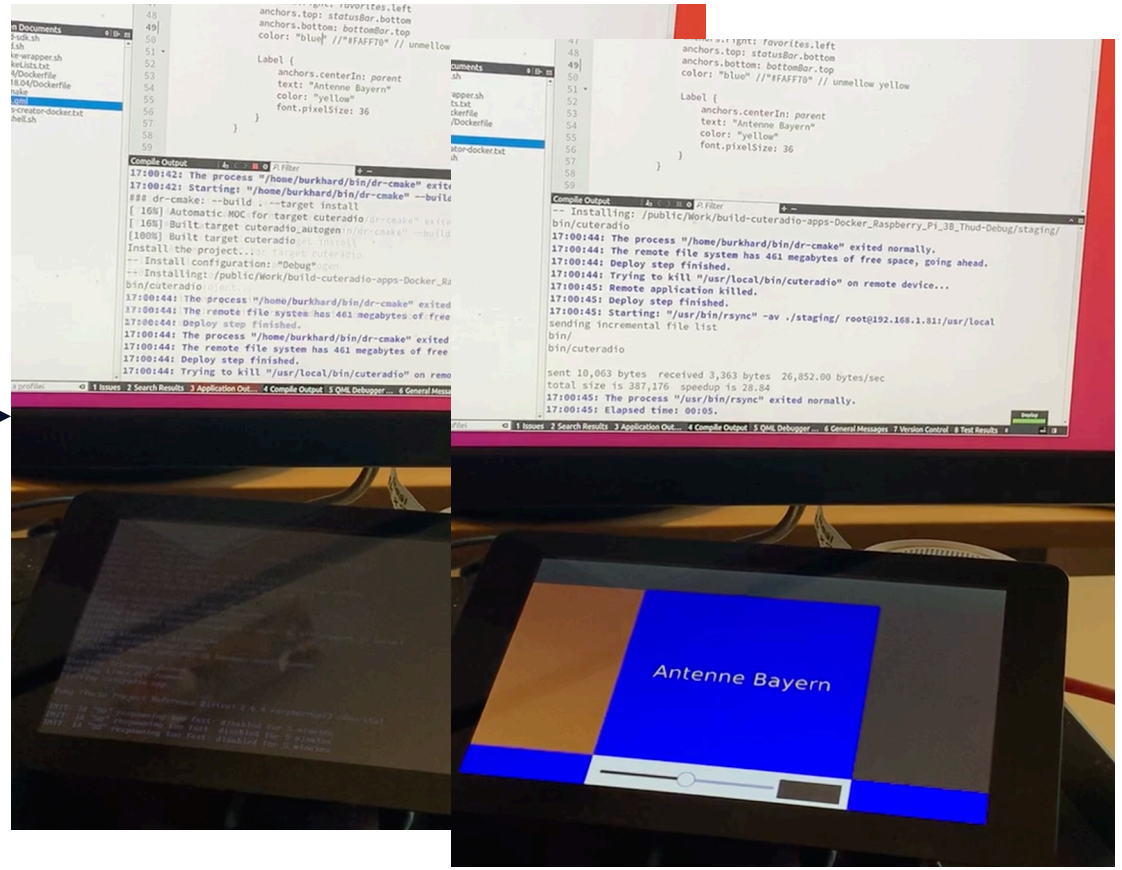
Running App on Device: Application Output



Running App on Device: Instead of a Video



Ctrl+R



References

[1] Docker Builds from QtCreator. Basis for this talk.

[2] Using Docker Containers for Yocto Builds. How to install Docker.

[3] Qt Embedded Systems – Part 1: Building a Linux Image with Yocto. First step to create a Qt SDK.

[4] Qt Embedded Systems – Part 2: Building a Qt SDK with Yocto. Creates the Qt SDK used in this talk.

[5] The Qt Company, Boot to Qt Software Stack. Official documentation of Boot2Qt.

[6] Tino Pyssysalo, Getting Started with Yocto & eLinux. How to install a Qt SDK and start developing an app.

Thank you ☺

Mail: burkhard.stubert@embeddeduse.com

Web: http://www.embeddeduse.com