Build and Run Embedded Apps Faster from QtCreator with Docker
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 make 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

How do we best build Apps from QtCreator?

All Qt SDKs built in Docker containers on same workstation
Solution 1 (Currently Used): Virtual Machines

- App A v3.5
  - Qt SDK
  - Yocto 2.7
  - Guest OS: Ubuntu 18.04
  - Host OS: Windows, MacOS, Linux

- App A v1.0
  - Qt SDK
  - Yocto 2.0
  - Guest OS: Ubuntu 14.04
  - Host OS: Windows, MacOS, Linux

- App B v2.3
  - Qt SDK
  - Yocto 2.4
  - Guest OS: Ubuntu 16.04
  - Host OS: Windows, MacOS, Linux

- App C v4.7
  - Qt SDK
  - Yocto 3.1
  - Guest OS: Ubuntu 19.10
  - Host OS: Windows, MacOS, Linux

- 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

- **Workstation:** Ubuntu 16.04

- **QtCreator**
  - **App A v3.5**
  - **Container**
  - **Qt SDK Yocto 2.7 Ubuntu 18.04**

- **QtCreator**
  - **App A v1.0**
  - **Container**
  - **Qt SDK Yocto 2.0 Ubuntu 14.04**

- **QtCreator**
  - **App B v2.3**
  - **Container**
  - **Qt SDK Yocto 2.4 Ubuntu 16.04**

- **QtCreator**
  - **App C v4.7**
  - **Container**
  - **Qt SDK Yocto 3.1 Ubuntu 19.10**

**Build details hidden in black box (container)**

**Reuse existing containers on fast workstation**
Solution 3 (Future): Containers on Remote Workstation

Apps developed on different computers

Apps built in containers on one or more remote workstations
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)

<table>
<thead>
<tr>
<th>Stage</th>
<th>Working Directory</th>
<th>CMake Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configure project</td>
<td>/tmp/QtCreator-Yqhjyl/qtc-cmake-MDHAJjOH =&gt; &lt;work-dir-1&gt;</td>
<td>cmake '-GUnix Makefiles' -C &lt;work-dir-1&gt;/qtcssettings.cmake</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/public/Work/cuteradio-apps</td>
</tr>
<tr>
<td>Generate build files</td>
<td>/public/Work/build-cuteradio-apps-Desktop_Qt_5_14_2_GCC_64bit-Debug =&gt; &lt;work-dir-2&gt;</td>
<td>cmake '-GUnix Makefiles' -C &lt;work-dir-2&gt;/qtcssettings.cmake</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/public/Work/cuteradio-app</td>
</tr>
<tr>
<td>Compile</td>
<td>&lt;work-dir-2&gt;</td>
<td>cmake --build . --target all -- -j4</td>
</tr>
<tr>
<td>Install</td>
<td>&lt;work-dir-2&gt;</td>
<td>cmake --build . --target install</td>
</tr>
</tbody>
</table>

*Call Docker wrapper instead of cmake*
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

Save script in $HOME/bin/dr-cmake

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

Resurrect single quotes removed by shell
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

- Workstation: Ubuntu 16.04
- QtCreator
- Cuteradio App
- Container
- Qt SDK for RPi 3
  - Yocto 2.6
  - Ubuntu 18.04

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

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:

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

In Docker container:

```bash
# 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

```bash
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"
...
```

Dockerfile

```bash
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"
...
```

Change all environment variables
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

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

![CMake Configuration in QtCreator](image-url)
Configuring QtCreator: Kit
Configuring QtCreator: Kit – CMake Generator

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiler:</td>
<td>&lt;NO compiler&gt;</td>
</tr>
<tr>
<td>Environment:</td>
<td></td>
</tr>
<tr>
<td>Debugger:</td>
<td></td>
</tr>
<tr>
<td>Qt version:</td>
<td></td>
</tr>
<tr>
<td>Qt mkspec:</td>
<td></td>
</tr>
<tr>
<td>Additional Qbs Profile Settings:</td>
<td></td>
</tr>
<tr>
<td>CMake Tool:</td>
<td></td>
</tr>
<tr>
<td>CMake generator:</td>
<td>CodeBlocks - Unix Makefiles, Platform: &lt;none&gt;, Toolset: &lt;none&gt;</td>
</tr>
<tr>
<td>CMake Configuration:</td>
<td>CMAKE_CXX_COMPILER:STRING=%{Compiler:Executable:Cxx}; ...</td>
</tr>
</tbody>
</table>

![Screenshot of QtCreator's CMake Generator settings window]
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

**Deployment**
- Method: Deploy to Remote Linux Host
- Files to deploy:
  - Override deployment data from build system
- Local File Path: Remote Directory

**Build**
- `dr cmake --build . --target install`
- Tool arguments:
  - Targets: Current executable, all, clean, install, test

**Check for free disk space**
- Remote path to check for free space: /
- Required disk space: SMB

**Kill current application instance**
- Custom Process Step: rsync -av /staging/ root@192.168.1.81:/usr/local
- Command: rsync
- Arguments: -av /staging/ root@192.168.1.81:/usr/local
- Working directory: %[buildDir]
Run Settings

**Run**

- **Run configuration**: Custom Executable (on Raspb
- **Remote executable**: `/usr/local/bin/cuteradio`
- **Local executable**: `%{CurrentProject:BuildPath}/staging/bin/cuteradio`
- **Command line arguments**: `-platform egls`
- **Working directory**: `/home/root`
- **Run in terminal**: [ ]
- **Forward to local display**: [ ]

**Environment**

- Use System Environment

---

29
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

cmake_install.cmake
-- Install configuration: "Debug"
-- Up-to-date: /public/Work/build-cuteradio-apps-Docker_Raspberry_Pi_3B_Thud-Debug/
staging/bin/cuteradio
18:49:43: The process "/home/burkhard/bin/dr-cmake" exited normally.
18:49:43: The remote file system has 461 megabytes of free space, going ahead.
18:49:43: Trying to kill "/usr/local/bin/cuteradio" on remote device...
18:49:44: Remote application killed.
18:49:44: Starting: "/usr/bin/rsync" -av ./staging/ root@192.168.1.81:/usr/local
sending incremental file list

sent 114 bytes received 13 bytes 84.67 bytes/sec
total size is 380,304 speedup is 2,994.52
18:49:45: The process "/usr/bin/rsync" exited normally.
18:49:45: Elapsed time: 00:05.
18:49:45: Starting /usr/local/bin/cuteradio -platform eglfs...
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'
Unable to query physical screen size, defaulting to 100 dpi.
To override, set QT_QPA_EGLFS_PHYSICAL_WIDTH and QT_QPA_EGLFS_PHYSICAL_HEIGHT (in millimeters).
@ Audio device = "default"
@ Audio device = "default:CARD=ALSA"
@ Audio device = "sysdefault:CARD=ALSA"
@ Default audio device = "default"
Running App on Device: Instead of a Video
References

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