Devcontainers and Embedded software development

Down the rabbit hole to never come back

What's the problem?

- All dev's need to install the tools
  - CubeIDE, MCUXpresso, MPLab, E2studio, SEGGER embedded...
  - arm-gcc v8/v9/v10, xc16...
  - other GNU tools: make, binutils, libraries, libc...
- 10 versions of gcc, Eclipse and other things installed
- Every time a new IDE to learn and customize

Can we solve it?

- Just install and configure the lot
- Remote development (e.g. over ssh)
- Combine all tools, libraries and the lot in one package
  - VM
  - Snap/Flatpack/AppImage
  - (Docker) container
  - Devcontainer with IDE supporting them

So... devcontainers... what are they?

containers.dev: *A development container (or dev container for short) allows you to use a **container** as a full-featured development environment. It can be used to run an application, to separate tools, libraries, or runtimes needed for working with a codebase, and to aid in continuous integration and testing. ...*

Down the rabbit hole: containers

*... you to use a **container** as a full-featured ...*

- Sandbox environment
- Like a VM: full OS
- Lightweight: uses kernel of the host
- Declarative: Infrastructure as Code
- Integration with lots of tools

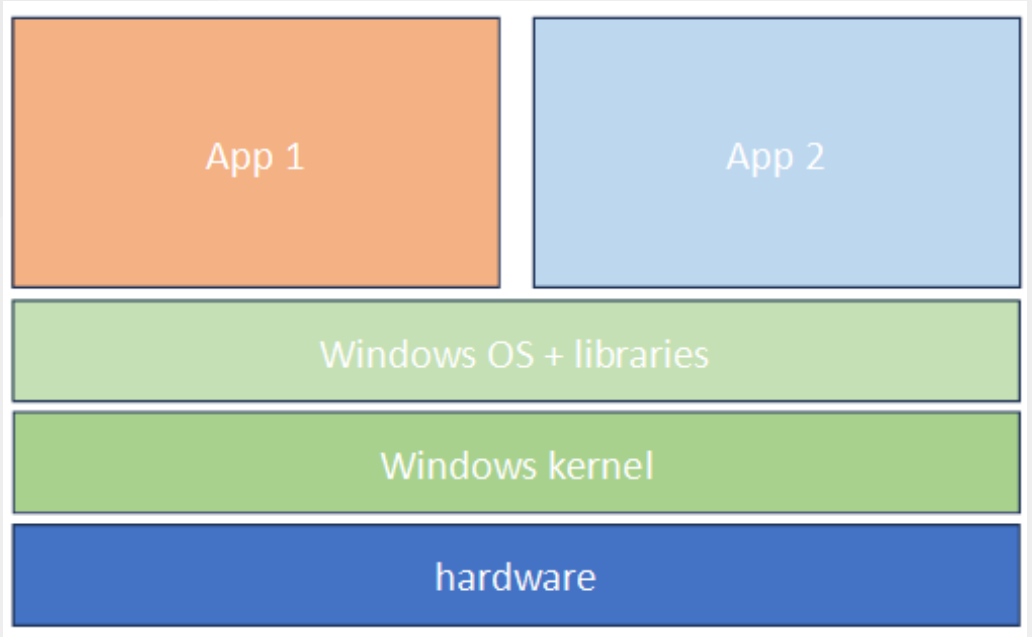
Down the rabbit hole: containers

They are used for:

- Easy deployment of (web) applications
- Micro-services on a (kubernetes) cluster
- Packaging of software and dependencies
  - running software locally
  - packaged software for use in CI/CD

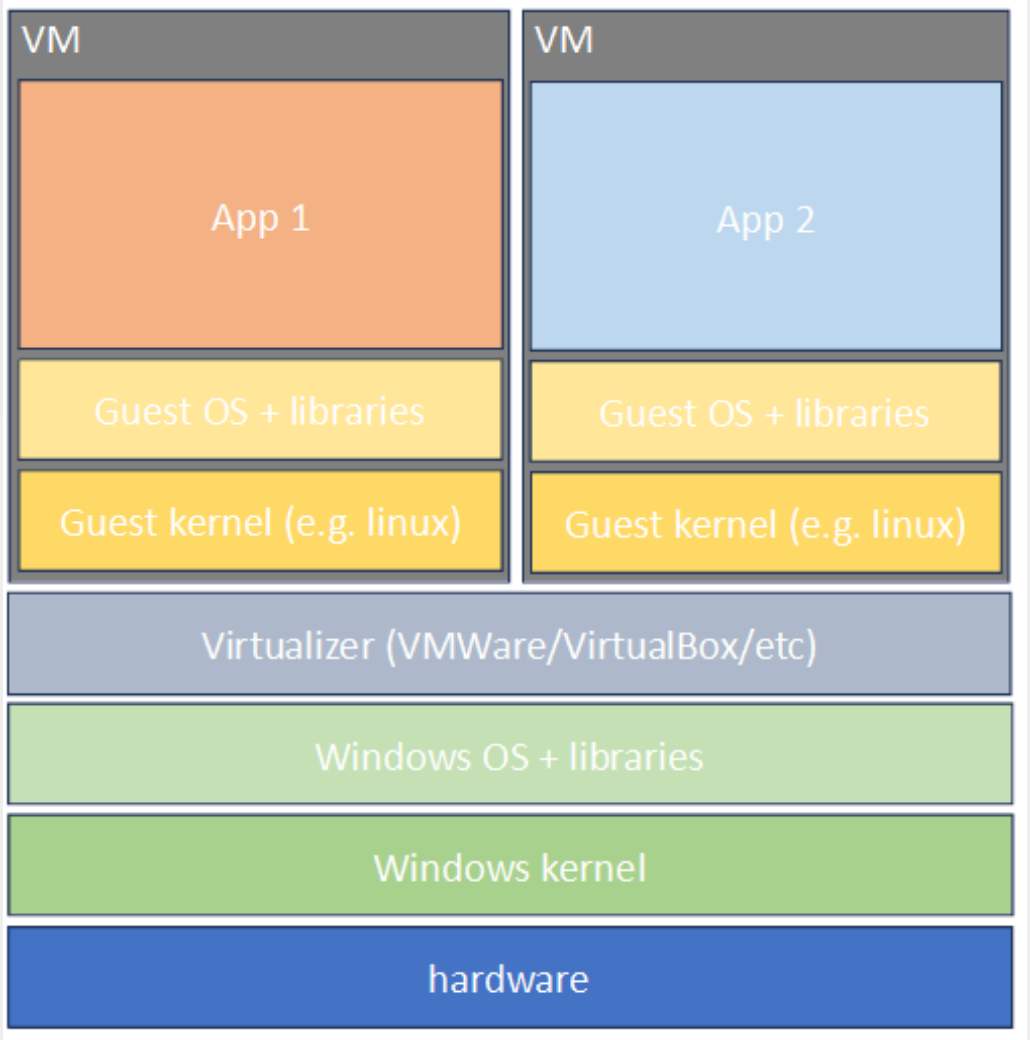
Like a VM and lightweight

Applications on top of OS



Like a VM and lightweight

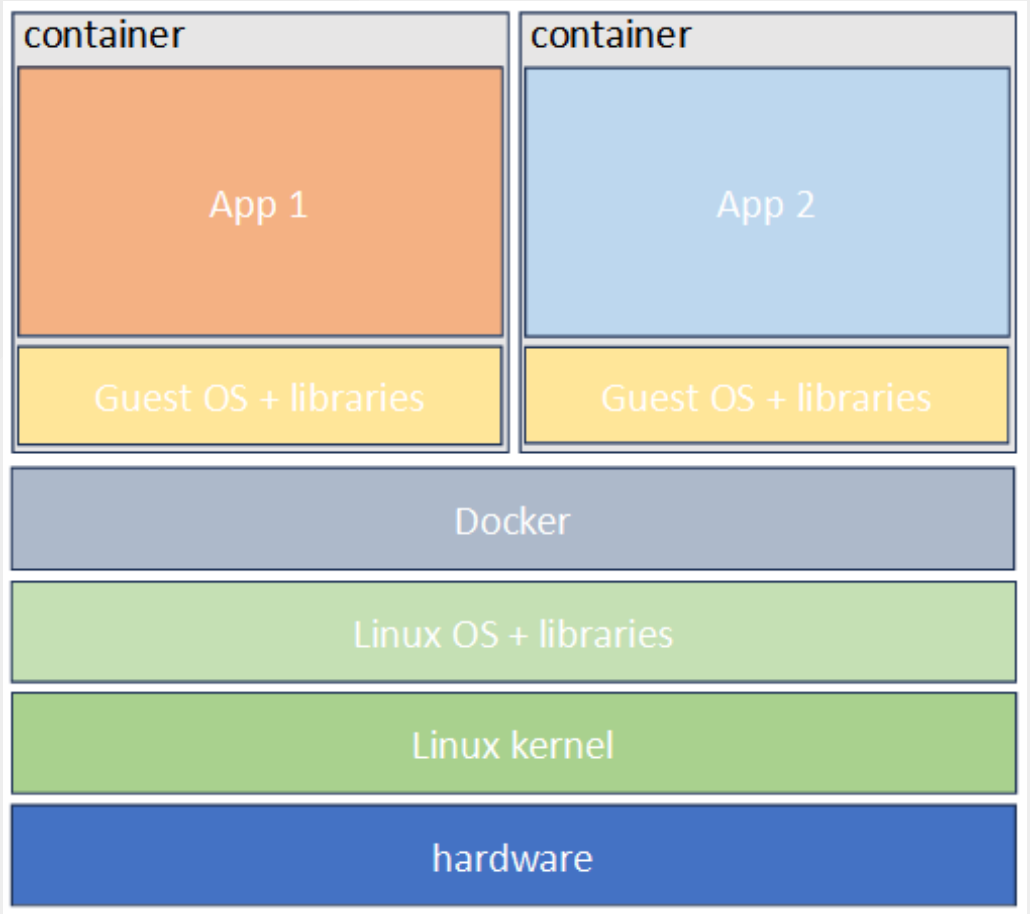
Applications in VM's



Like a VM and lightweight

Applications in Docker containers

Alpine: *... A container requires no more than **8 MB** ...*



Declarative: Infrastructure as Code

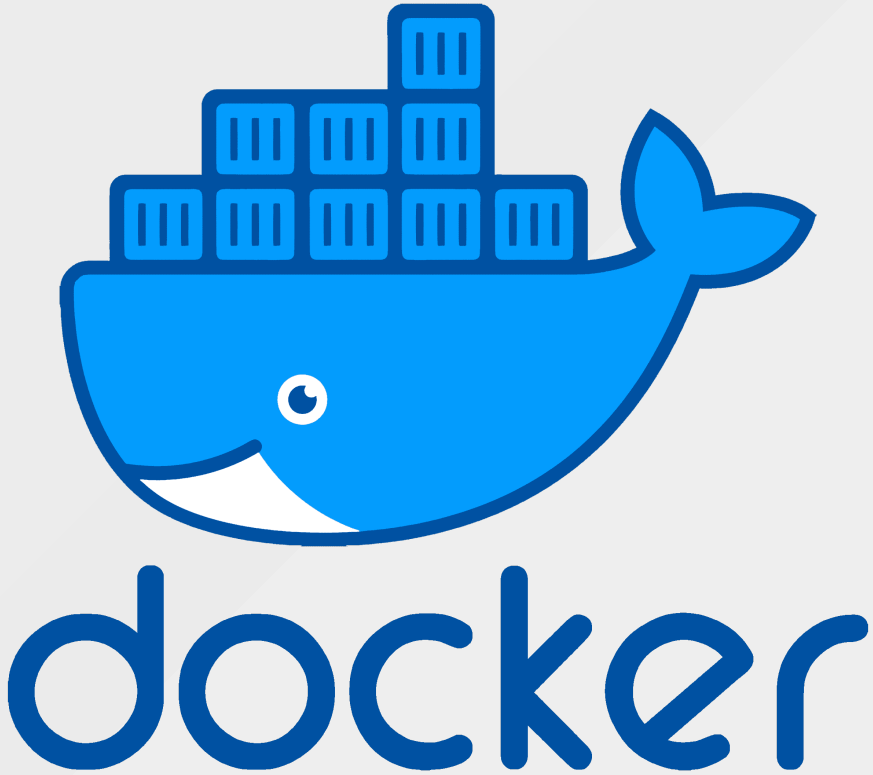
```
FROM debian
ENV DEBIAN_FRONTEND=noninteractive

RUN apt update -y && apt install -y \
    build-essential \
    cmake

RUN useradd -ms /bin/bash someuser
WORKDIR /home/someuser
USER someuser

CMD /bin/bash
```

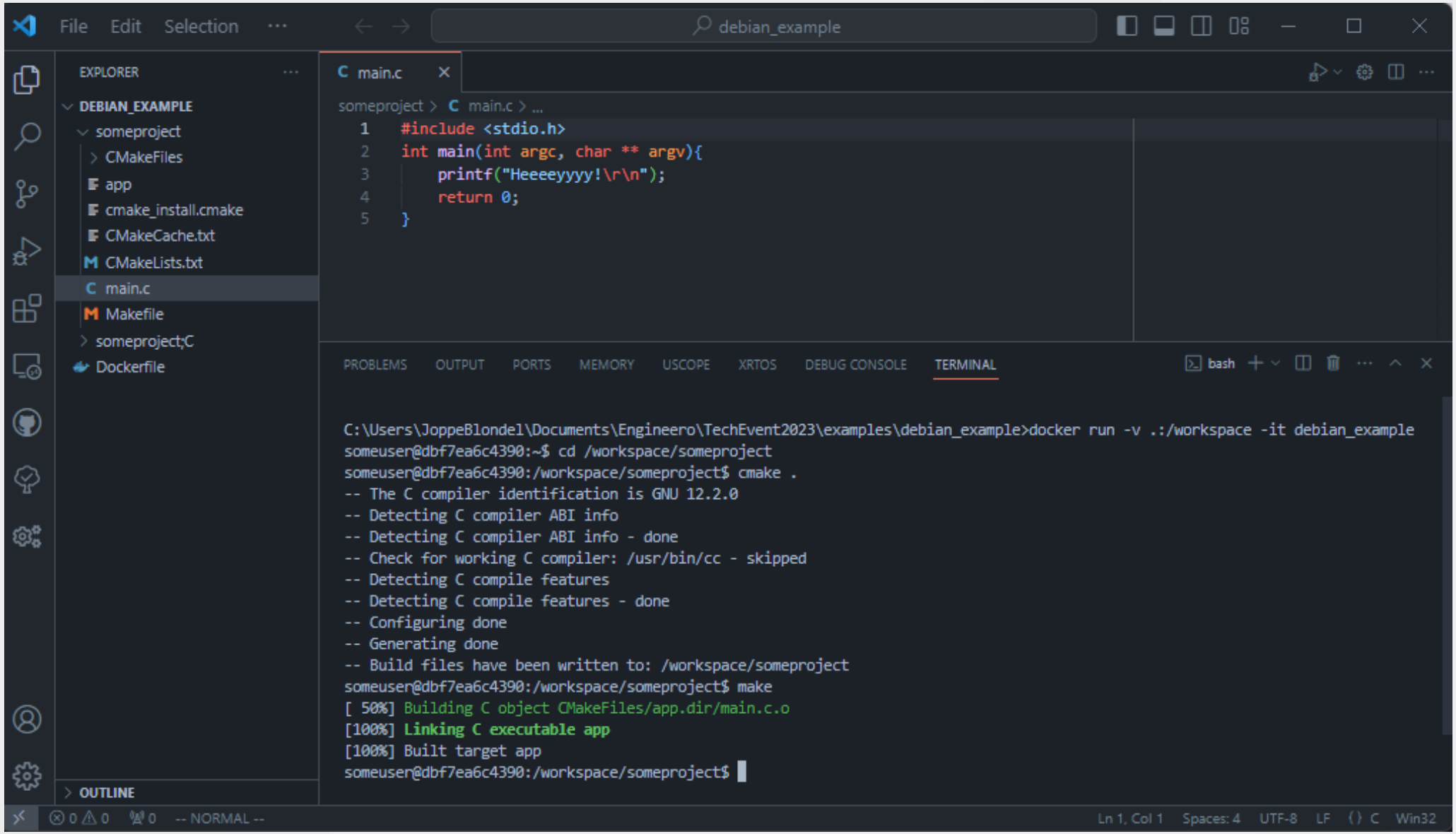
Declarative: Infrastructure as Code

- To create a container image:
  `docker build --tag 'debian_example' .`
- To run a container image:
  `docker run -it 'debian_example'`
- This will get you a Debian shell

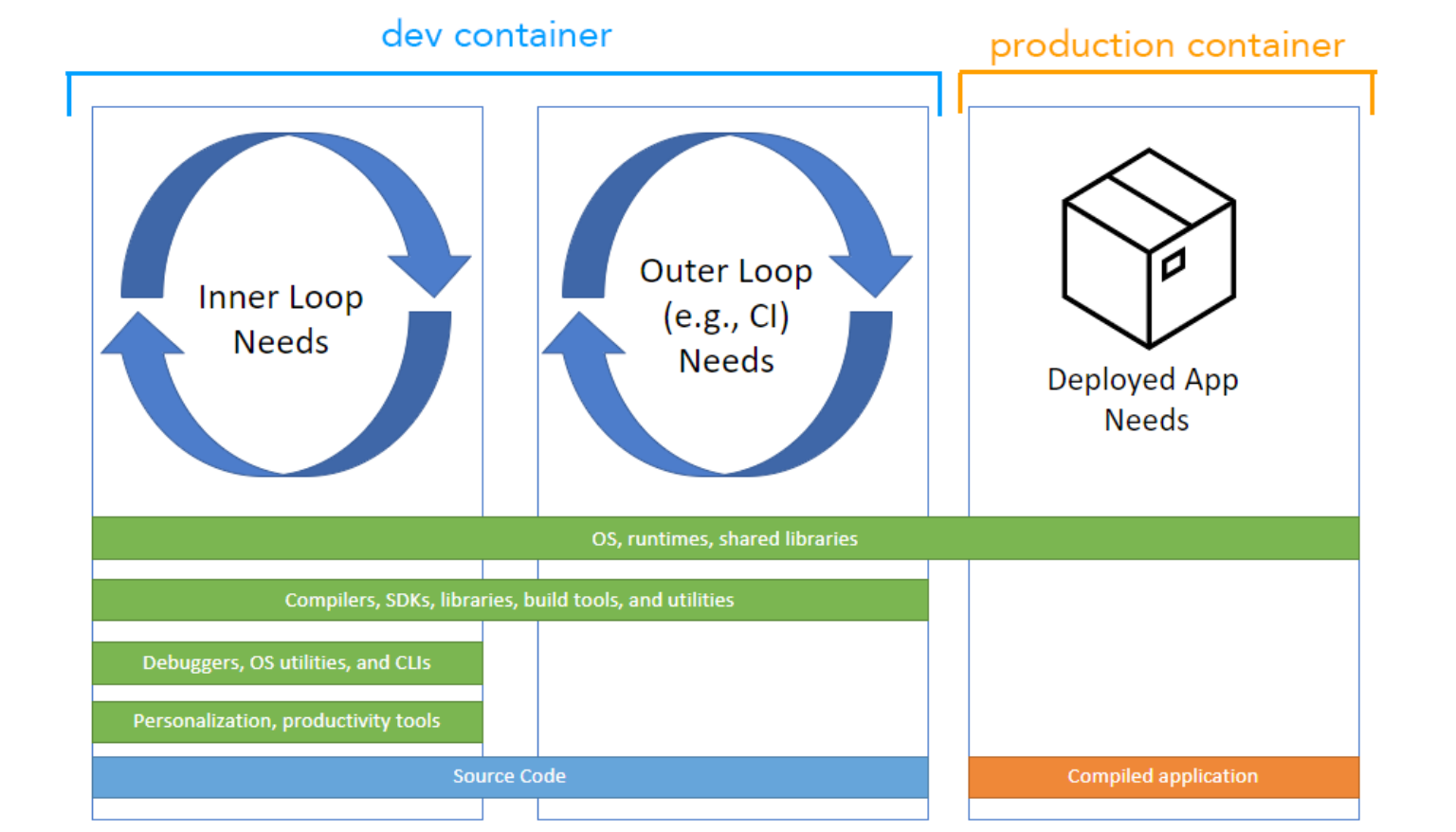
```
$ docker run -it debian_example
someuser@e9bf5806d588:~$ pwd
/home/someuser
someuser@e9bf5806d588:~$
```

Containers and Docker




File Edit Selection
debian_example
EXPLORER
DEBIAN_EXAMPLE
someproject
CMakeFiles
app
cmake_install.cmake
CMakeCache.txt
CMakeLists.txt
main.c
Makefile
someproject;C
Dockerfile
OUTLINE
main.c
someproject > main.c > ...
1 #include <stdio.h>
2 int main(int argc, char ** argv){
3 printf("Heeeeyyyy!\r\n");
4 return 0;
5 }
PROBLEMS OUTPUT PORTS MEMORY USCOPE XRTOS DEBUG CONSOLE TERMINAL
bash
C:\Users\JoppeBlondel\Documents\Engineero\TechEvent2023\examples\debian_example>docker run -v .:/workspace -it debian_example
someuser@dbf7ea6c4390:~$ cd /workspace/someproject
someuser@dbf7ea6c4390:/workspace/someproject$ cmake .
-- The C compiler identification is GNU 12.2.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /workspace/someproject
someuser@dbf7ea6c4390:/workspace/someproject$ make
[50%] Building C object CMakeFiles/app.dir/main.c.o
[100%] Linking C executable app
[100%] Built target app
someuser@dbf7ea6c4390:/workspace/someproject$
-- NORMAL --
Ln 1, Col 1 Spaces: 4 UTF-8 LF C Win32

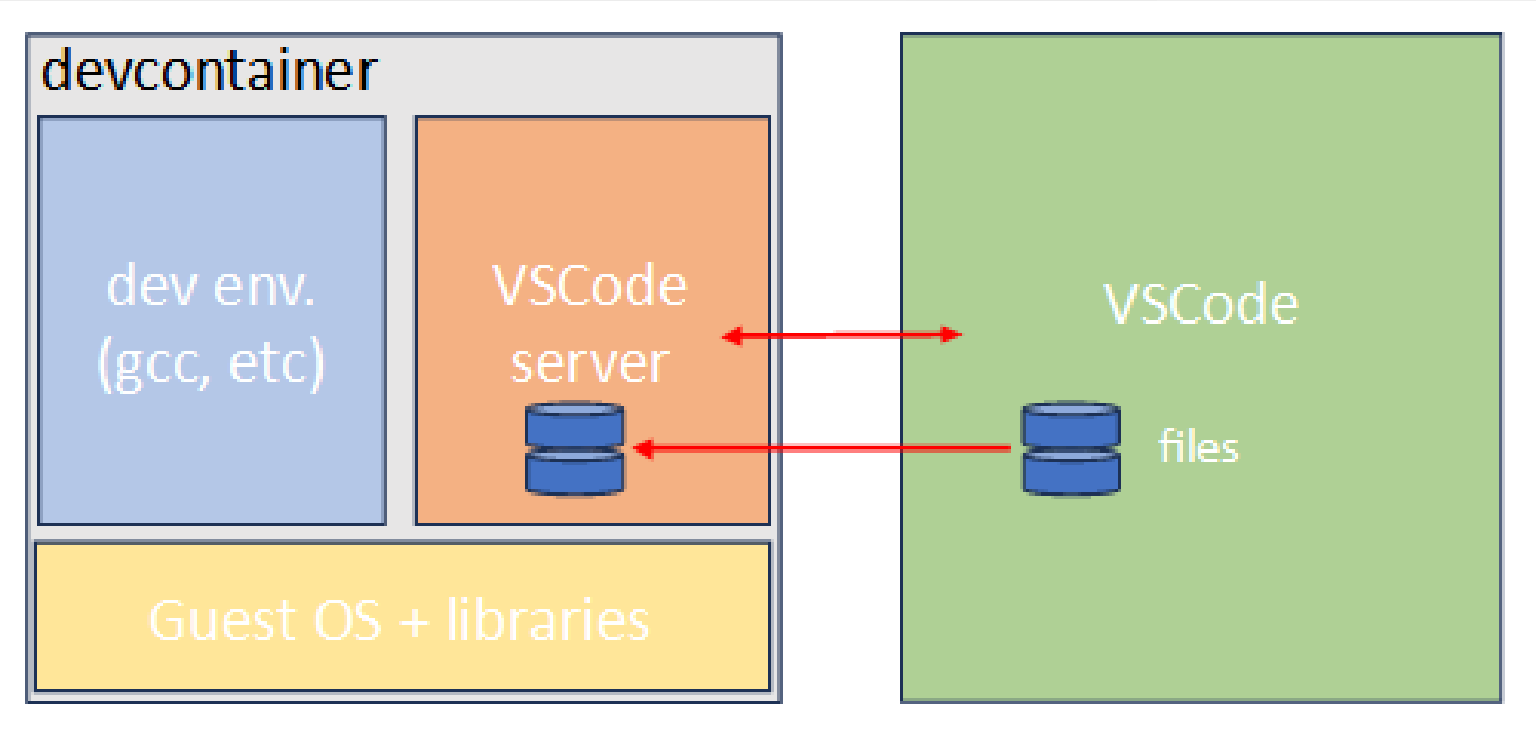
Devcontainers: containers with spice

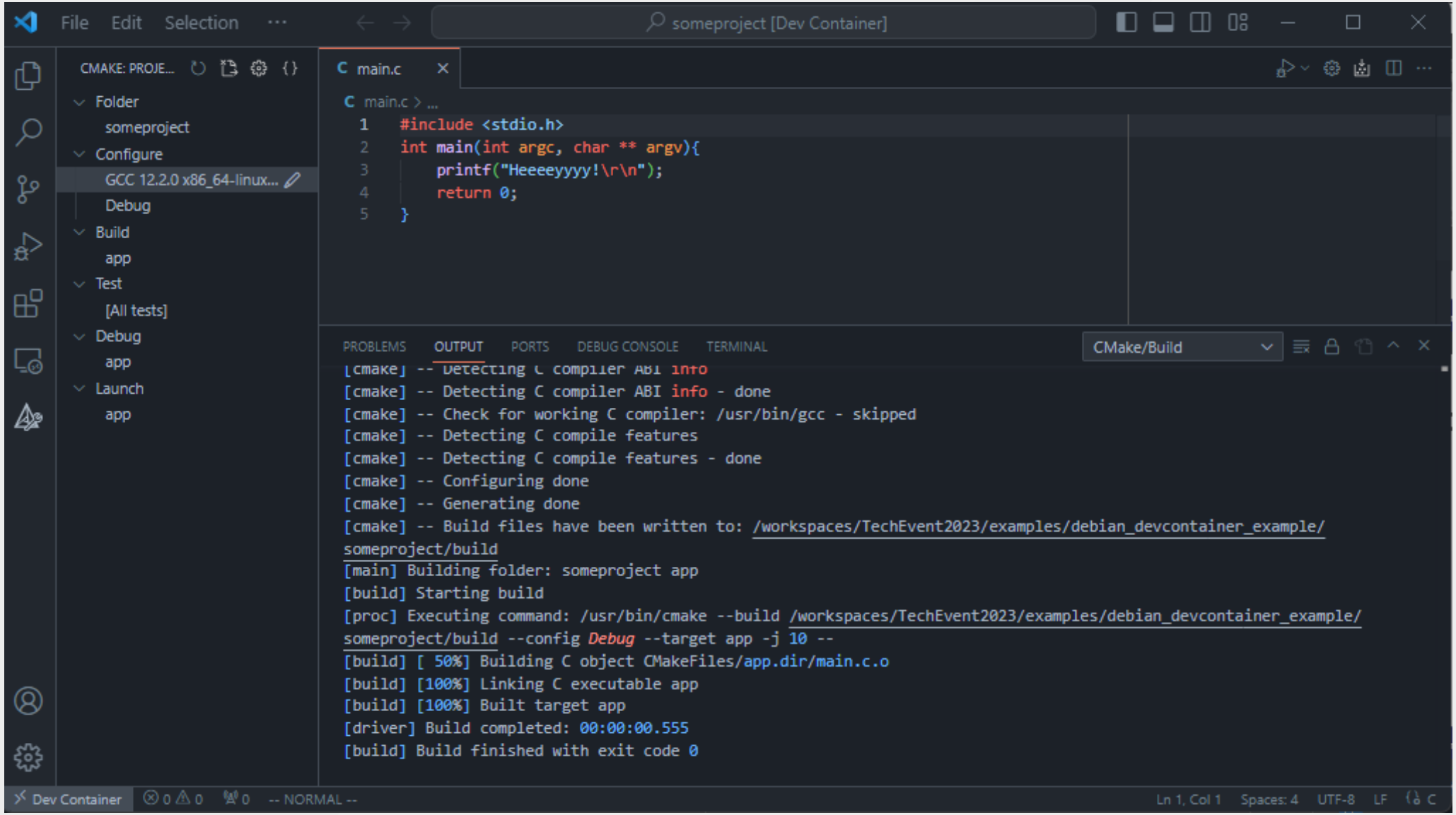


Devcontainers: containers with spice

- Full development environment
- Utilities and personalization
- Configuration of IDE

Devcontainers and VSCode




File Edit Selection
someproject [Dev Container]
CMAKE: PROJE...
Folder
someproject
Configure
GCC 12.2.0 x86_64-linux...
Debug
Build
app
Test
[All tests]
Debug
app
Launch
app
main.c
#include <stdio.h>
int main(int argc, char ** argv){
printf("Heeeeyyyy!\r\n");
return 0;
}
PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL
CMake/Build
[cmake] -- Detecting C compiler ABI info
[cmake] -- Detecting C compiler ABI info - done
[cmake] -- Check for working C compiler: /usr/bin/gcc - skipped
[cmake] -- Detecting C compile features
[cmake] -- Detecting C compile features - done
[cmake] -- Configuring done
[cmake] -- Generating done
[cmake] -- Build files have been written to: /workspaces/TechEvent2023/examples/debian_devcontainer_example/someproject/build
[main] Building folder: someproject app
[build] Starting build
[proc] Executing command: /usr/bin/cmake --build /workspaces/TechEvent2023/examples/debian_devcontainer_example/someproject/build --config Debug --target app -j 10 --
[build] [50%] Building C object CMakeFiles/app.dir/main.c.o
[build] [100%] Linking C executable app
[build] [100%] Built target app
[driver] Build completed: 00:00:00.555
[build] Build finished with exit code 0
Dev Container
-- NORMAL --
Ln 1, Col 1
Spaces: 4
UTF-8
LF
C

Devcontainers: an example

```
{
    "build": { "dockerfile" : "Dockerfile" },
    "customizations": {
        "vscode": {
            "extensions" : [
                "ms-vscode.cpptools-extension-pack",
                "ms-vscode.cpptools",
                "ms-vscode.cmake-tools",
            ]
        }
    }
}
```

Devcontainers and VSCode

- Seamless: functions as local instance
- Executes build tools, debugger and other tools from container
- VSCode Configuration and plugins declared in the json file
- Make sure to install the Dev Containers plugin

  `ms-vscode-remote.remote-containers`

The full story? Nope...

We develop firmware, not normal applications or web-apps

The full story? Nope...

We develop firmware, not normal applications or web-apps

... So I lied?

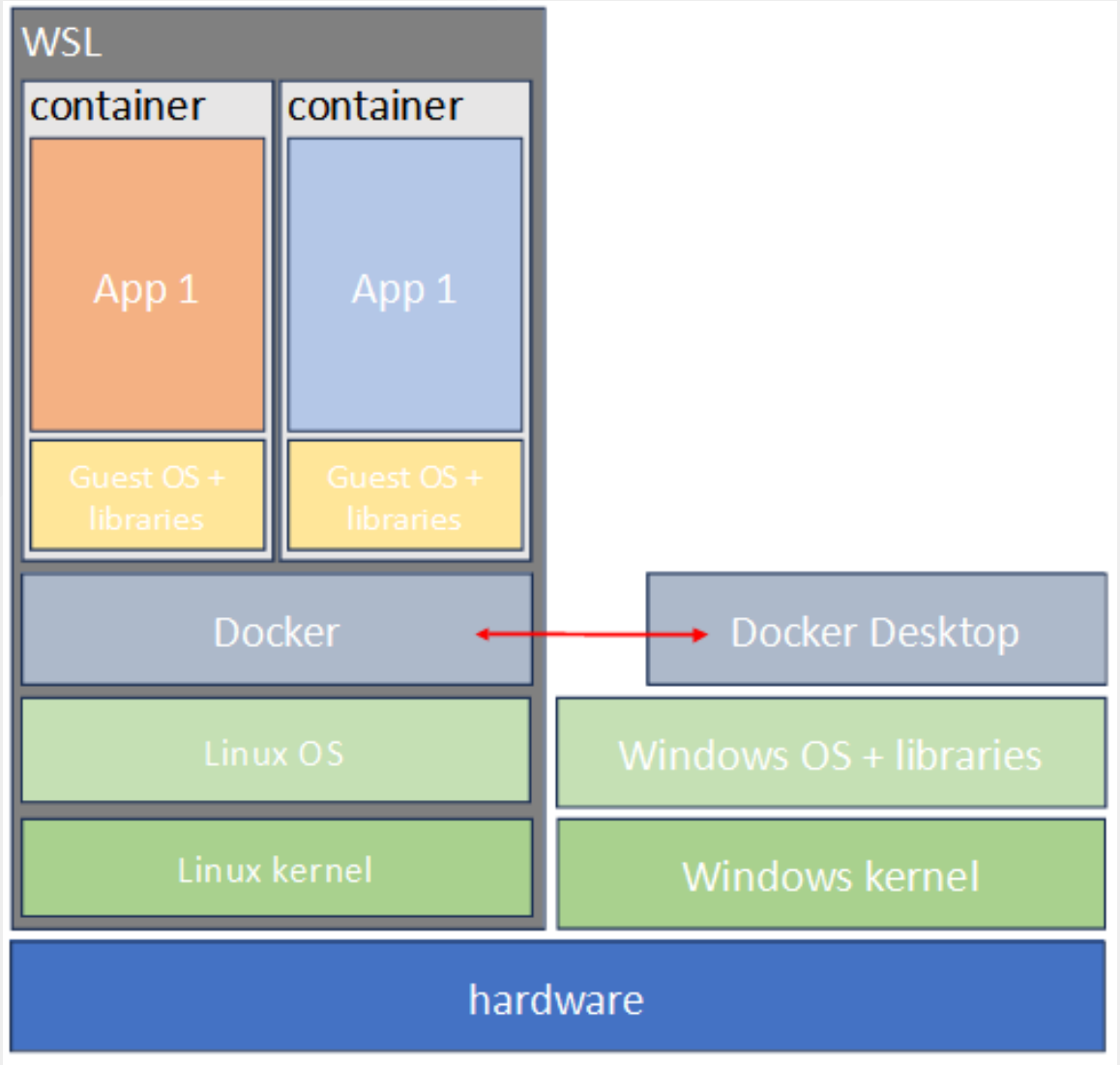
Embedded development

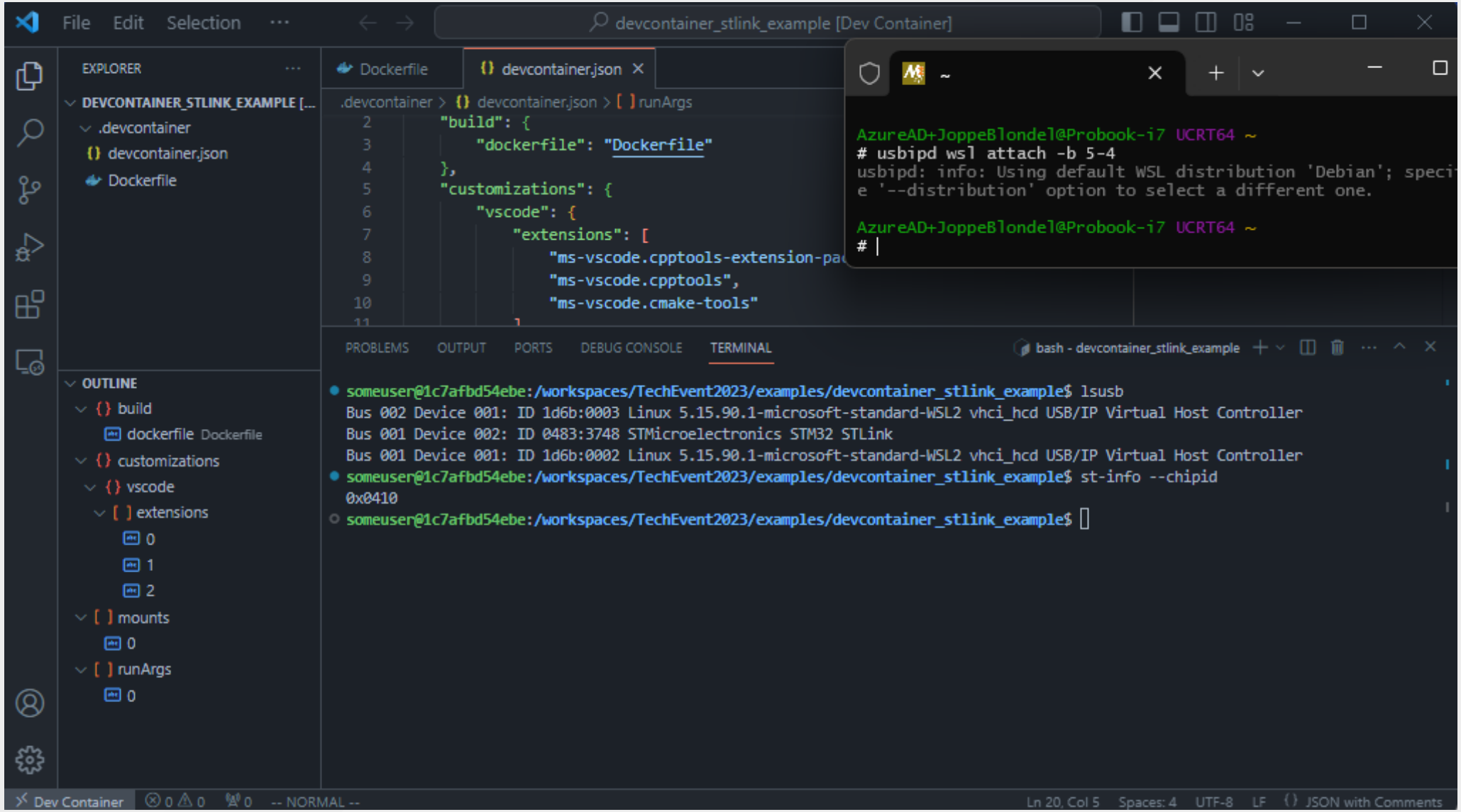
- Debugging of non-native applications
- Use of physical debugger (e.g. JLink/STLink/BMP)
- **we need USB**
  - so just pass the USB to the container?
  - .....

Docker on Windows

Docker is in it's core a linux tool

- Containers themselves on WSL
- USB to WSL: USBIP
  - see readme




File Edit Selection
devcontainer_stlink_example [Dev Container]
EXPLORER
DEVCONTAINER_STLINK_EXAMPLE [...
.devcontainer
devcontainer.json
Dockerfile
Dockerfile
devcontainer.json
.devcontainer > devcontainer.json > [ ] runArgs
"build": {
"dockerfile": "Dockerfile"
},
"customizations": {
"vscode": {
"extensions": [
"ms-vscode.cpptools",
"ms-vscode.cmake-tools"
~
AzureAD+JoppeBlondel@Probook-i7 UCRT64 ~
usbipd wsl attach -b 5-4
usbipd: info: Using default WSL distribution 'Debian'; specify
e '--distribution' option to select a different one.
AzureAD+JoppeBlondel@Probook-i7 UCRT64 ~
#
OUTLINE
build
dockerfile Dockerfile
customizations
vscode
extensions
0
1
2
mounts
0
runArgs
0
PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL
bash - devcontainer_stlink_example
someuser@1c7afbd54ebe:/workspaces/TechEvent2023/examples/devcontainer_stlink_example$ lsusb
Bus 002 Device 001: ID 1d6b:0003 Linux 5.15.90.1-microsoft-standard-WSL2 vhci_hcd USB/IP Virtual Host Controller
Bus 001 Device 002: ID 0483:3748 STMicroelectronics STM32 STLink
Bus 001 Device 001: ID 1d6b:0002 Linux 5.15.90.1-microsoft-standard-WSL2 vhci_hcd USB/IP Virtual Host Controller
someuser@1c7afbd54ebe:/workspaces/TechEvent2023/examples/devcontainer_stlink_example$ st-info --chipid
0x0410
someuser@1c7afbd54ebe:/workspaces/TechEvent2023/examples/devcontainer_stlink_example$
Dev Container
-- NORMAL --
Ln 20, Col 5
Spaces: 4
UTF-8
LF
JSON with Comments

All together

- Toolchain, debugging tools and IDE configurations packaged together
- Declarative
- Lightweight
- Multi-purpose: development and CI/CD