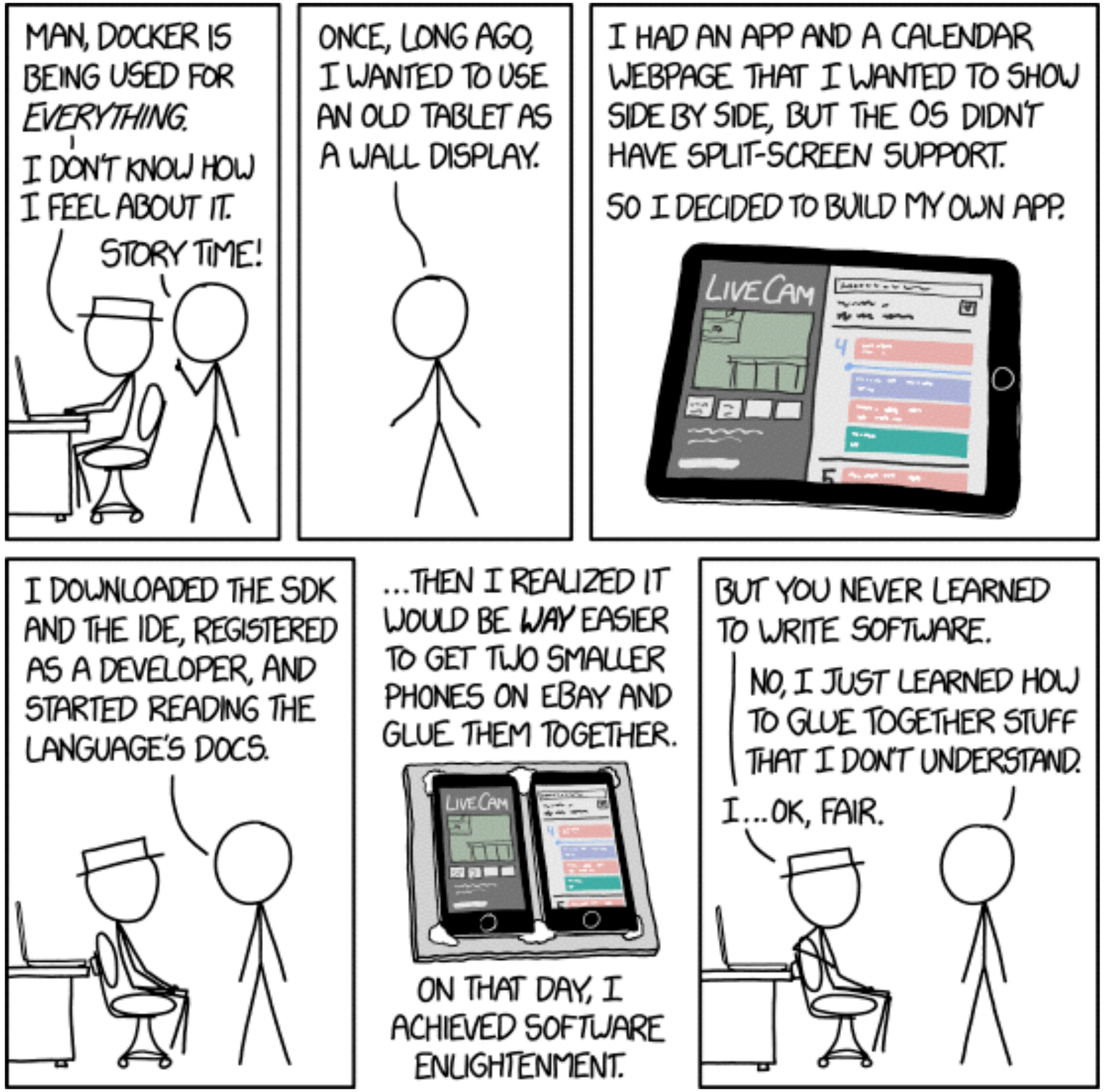
C++NOW 2018 JASON RICE

DOCKER BASED C++

DEPENDENCY AND BUILD MANAGEMENT


MAN, DOCKER IS BEING USED FOR EVERYTHING.
I DON'T KNOW HOW I FEEL ABOUT IT.
STORY TIME!
ONCE, LONG AGO, I WANTED TO USE AN OLD TABLET AS A WALL DISPLAY.
I HAD AN APP AND A CALENDAR WEBPAGE THAT I WANTED TO SHOW SIDE BY SIDE, BUT THE OS DIDN'T HAVE SPLIT-SCREEN SUPPORT.
SO I DECIDED TO BUILD MY OWN APP.
LIVE CAM
I DOWNLOADED THE SDK AND THE IDE, REGISTERED AS A DEVELOPER, AND STARTED READING THE LANGUAGE'S DOCS.
...THEN I REALIZED IT WOULD BE WAY EASIER TO GET TWO SMALLER PHONES ON EBAY AND GLUE THEM TOGETHER.
LIVE CAM
ON THAT DAY, I ACHIEVED SOFTWARE ENLIGHTENMENT.
BUT YOU NEVER LEARNED TO WRITE SOFTWARE.
NO, I JUST LEARNED HOW TO GLUE TOGETHER STUFF THAT I DON'T UNDERSTAND.
I...OK, FAIR.

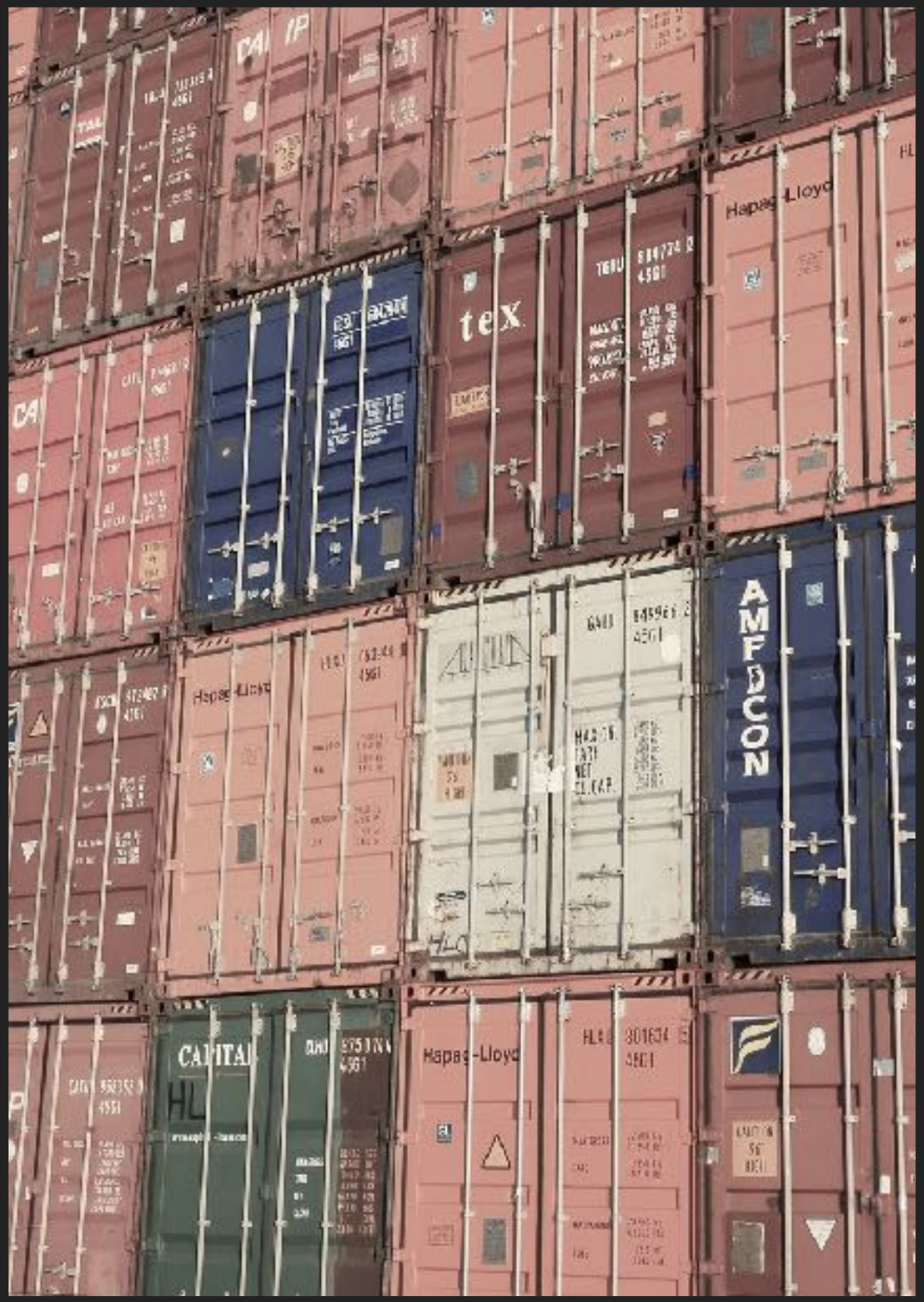
xkcd

OVERVIEW

- Images and Containers
- Dockerfile
- Hello World
- Multi-Stage Build
- Build a Toolchain
- Contributing to an Open Source Project
- CppDock

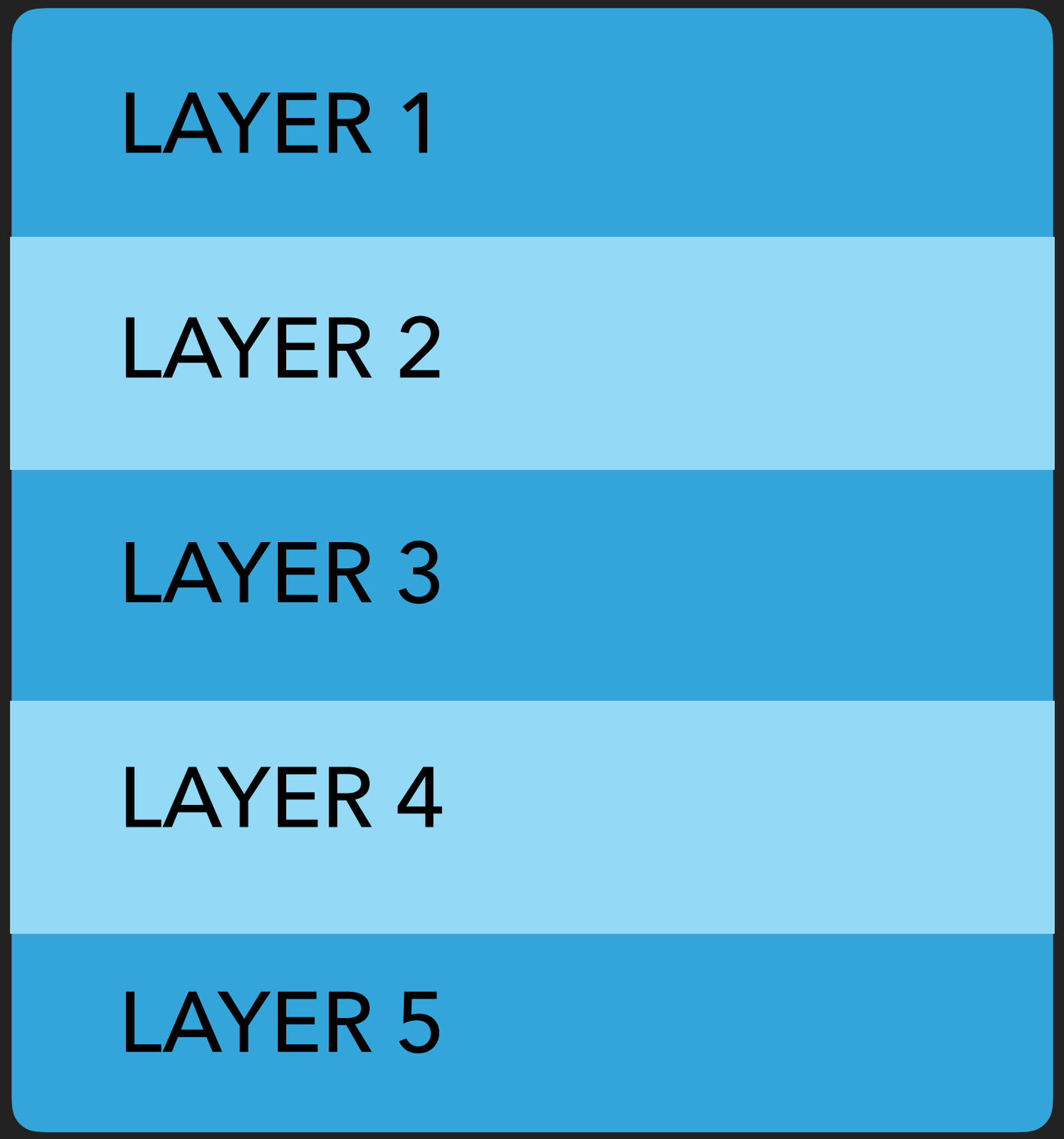
WHAT IS DOCKER?

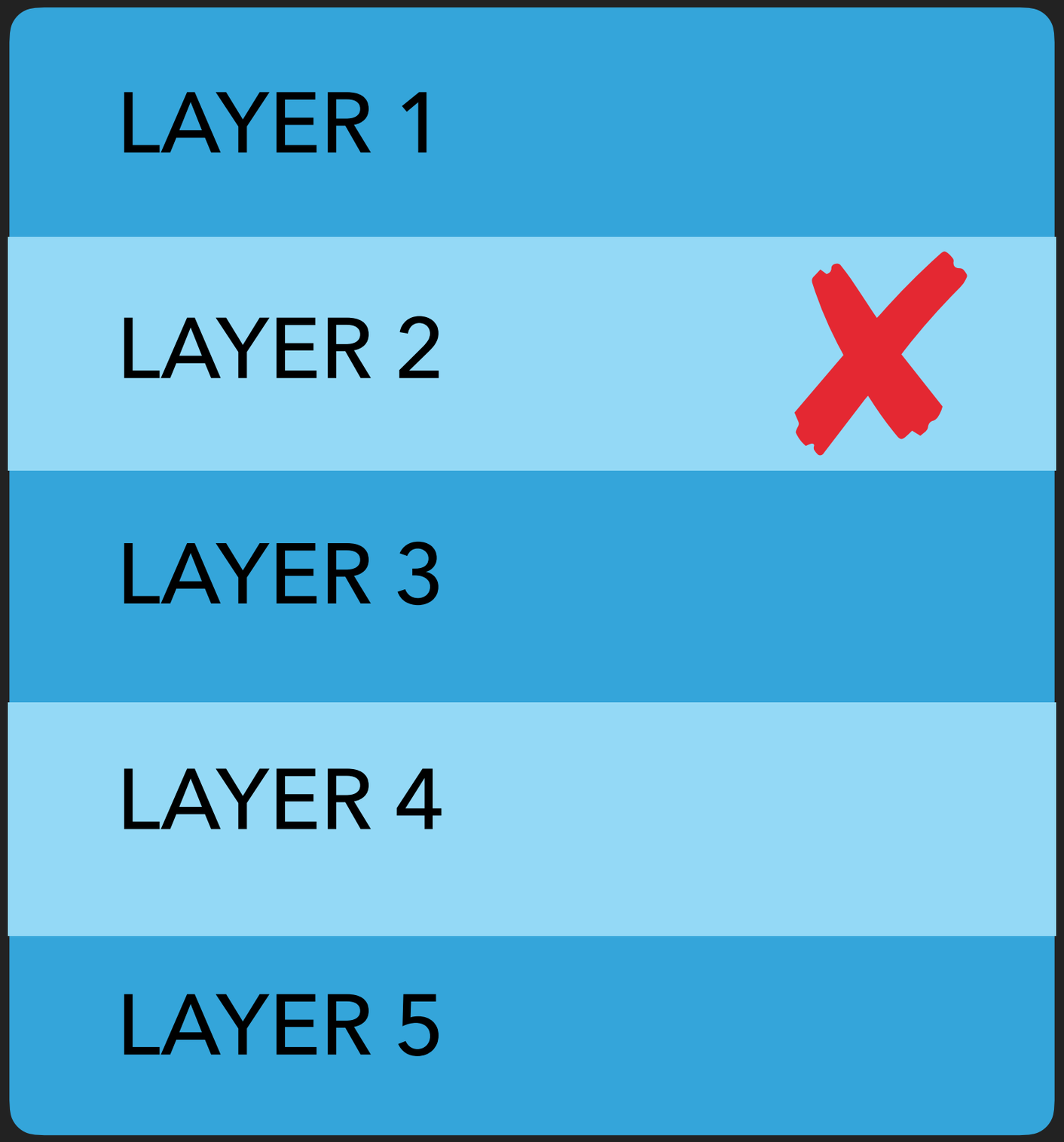
- Open platform
- Creates and runs containers
- Each container is an isolated system.
- Fast startup time
- Low execution overhead vs VM
- Reproducible builds
- Host images in registry
- Connectable, deployable, and it can even be used as a process manager

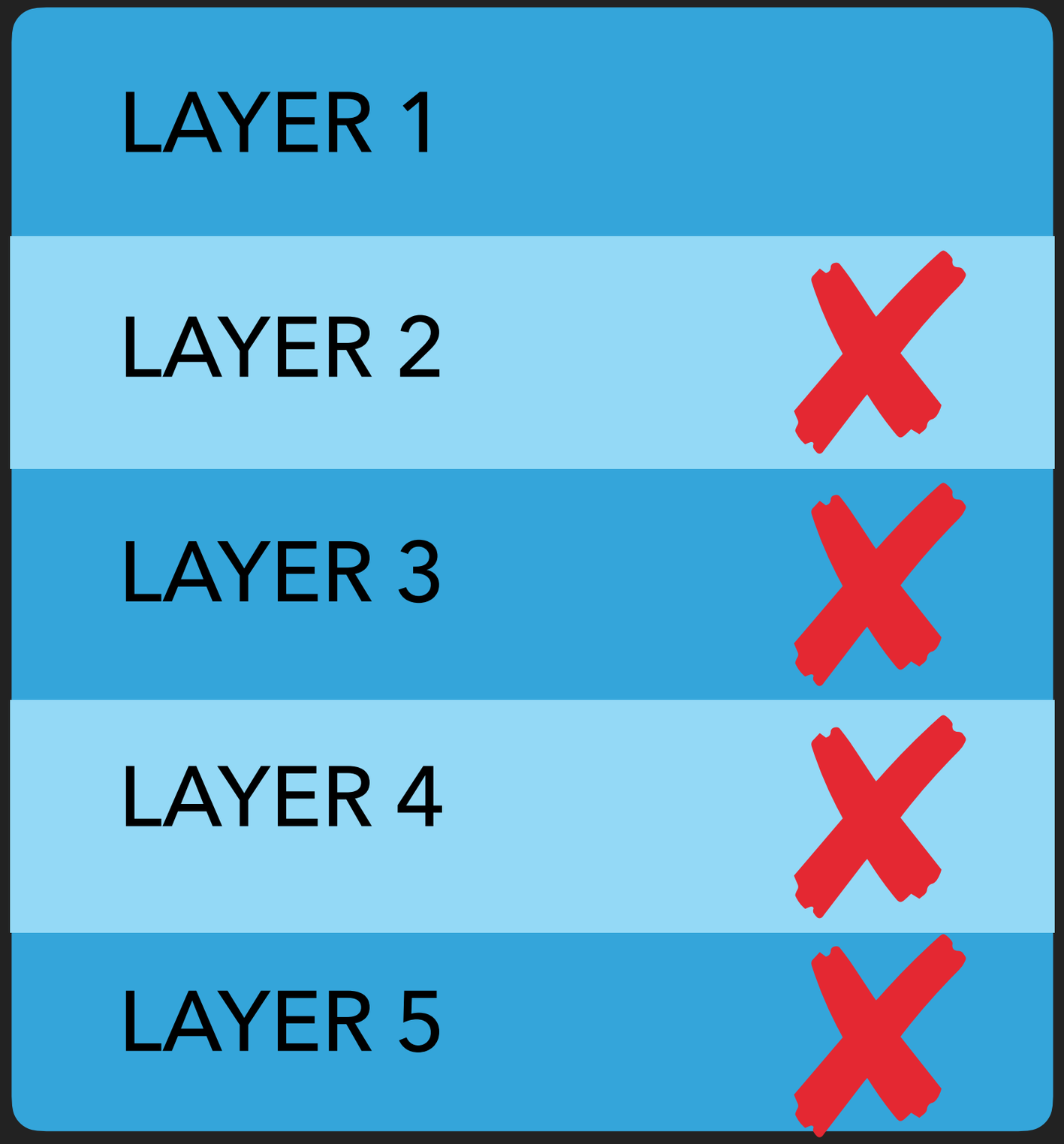


DOCKER IMAGES

- Multiple layers on a union filesystem
- Images are immutable
- Each layer is cached for a given input
- Layers are built in series
- Multi-stage builds can build layers independently
- Files use copy-on-write when modified in subsequent layer
- Created using `docker build`

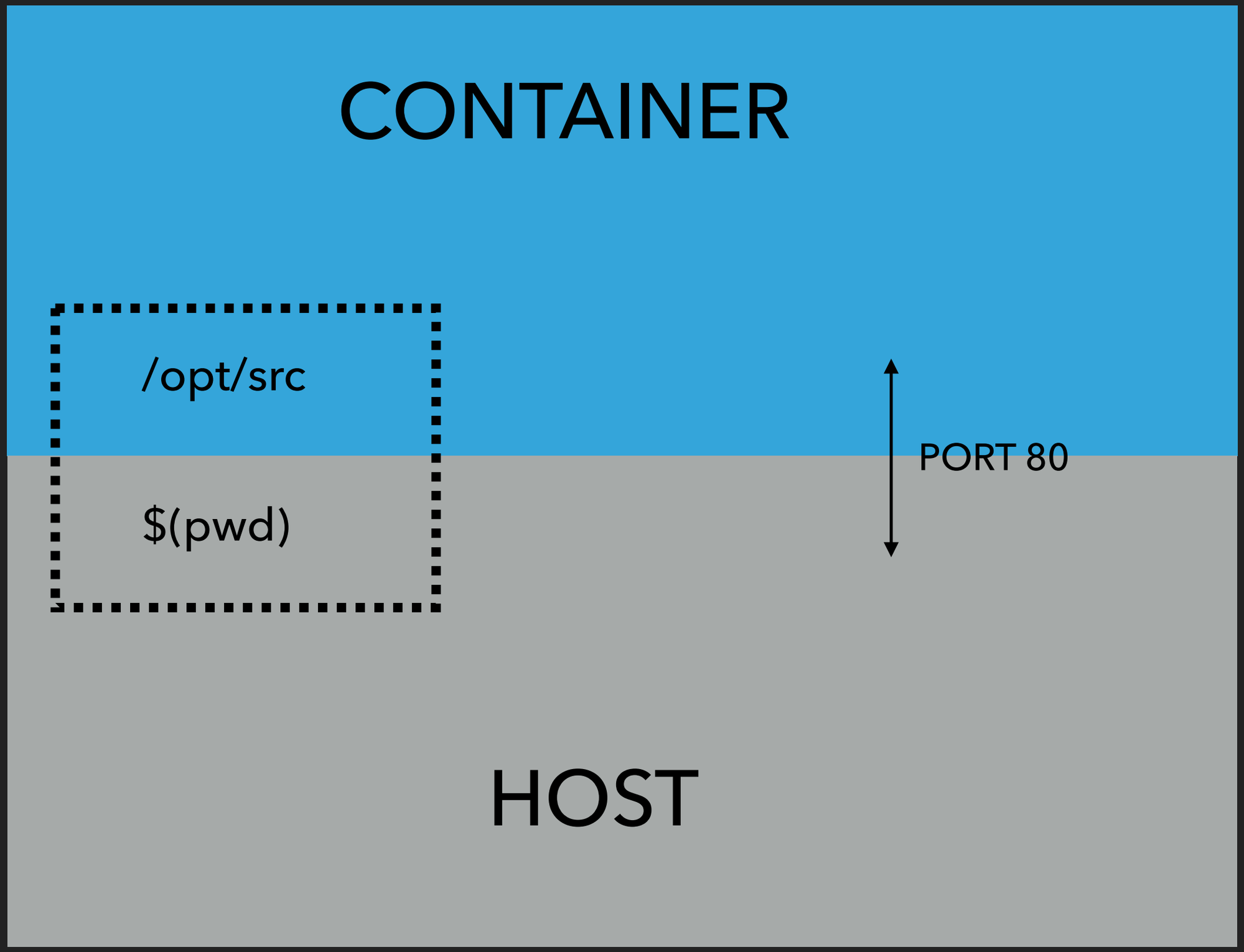

LAYER 1
LAYER 2
LAYER 3
LAYER 4
LAYER 5


LAYER 1
LAYER 2
LAYER 3
LAYER 4
LAYER 5


LAYER 1
LAYER 2
LAYER 3
LAYER 4
LAYER 5

DOCKER CONTAINERS

- A runnable instance of an Image
- Adds a final mutable layer
- Low overhead via CoW strategy (32k)
- Can create mount points on host filesystem
- Composable to create an ad hoc, private network
- Exportable to raw file system or VM
- Run with `docker run`


CONTAINER
/opt/src
$(pwd)
PORT 80
HOST

DOCKERFILE

- Documentative DSL to define steps to build an image
- Defines a base image layer
- Defines Layers that can copy files into the image being built or run shell commands in the image's context
- Specifies default env vars, ports, volumes, and a shell command to run at execution time

```
FROM debian:stretch-slim

RUN dpkg --add-architecture i386

RUN apt-get update && apt-get install -y --no-install-recommends \
        gcc \
        libc6-dev \
        make \
        \
        libc6-dev:i386 \
        libgcc-6-dev:i386 \
        \
        libc6-dev-arm64-cross \
        libc6-dev-armel-cross \
        libc6-dev-armhf-cross \
        libc6-dev-ppc64el-cross \
        libc6-dev-s390x-cross \
        \
        gcc-aarch64-linux-gnu \
        gcc-arm-linux-gnueabi \
        gcc-arm-linux-gnueabihf \
        gcc-powerpc64le-linux-gnu \
        gcc-s390x-linux-gnu \
        \
        file \
    && rm -rf /var/lib/apt/lists/*

WORKDIR /usr/src/hello
COPY . .

RUN set -ex; \
    make clean all test \
        TARGET_ARCH='amd64' \
        CC='x86_64-linux-gnu-gcc' \
        STRIP='x86_64-linux-gnu-strip'

RUN set -ex; \
    make clean all \
        TARGET_ARCH='arm32v5' \
        CC='arm-linux-gnueabi-gcc' \
        STRIP='arm-linux-gnueabi-strip'

RUN set -ex; \
    make clean all \
        TARGET_ARCH='arm32v7' \
        CC='arm-linux-gnueabihf-gcc' \
        STRIP='arm-linux-gnueabihf-strip'

RUN set -ex; \
    make clean all \
```

HELLO WORLD C++ STYLE

```
#include <iostream>

int main() {
    std::cout << "Hello, world!\n";
}
```

HELLO WORLD C++ STYLE – FILE LAYOUT

```
~/cppnow18_docker_cpp/example/hello_world$ ls
Dockerfile  main.cpp
```

HELLO WORLD C++ STYLE - DOCKERFILE

```
FROM ricejasonf/cppdock:linux_x64

COPY main.cpp /opt/build/

WORKDIR /opt/build

RUN clang++ -stdlib=libc++ -lc++abi main.cpp

CMD ["./a.out"]
```

HELLO WORLD C++ STYLE - BUILD

```
~/cppnow18_docker_cpp/example/hello_world$ docker build -t hello_world .
Sending build context to Docker daemon  3.072kB
Step 1/5 : FROM ricejasonf/cppdock:linux_x64
 ---> 78b2d874d92c
Step 2/5 : COPY main.cpp /opt/build/
 ---> 2004574680e6
Step 3/5 : WORKDIR /opt/build
Removing intermediate container 6ac6b1d93c0b
 ---> e1fe223d84da
Step 4/5 : RUN clang++ -stdlib=libc++ -lc++abi main.cpp
 ---> Running in b5ccc39efc24
Removing intermediate container b5ccc39efc24
 ---> c6c593a12857
Step 5/5 : CMD ["./a.out"]
 ---> Running in 6c243b494d7f
Removing intermediate container 6c243b494d7f
 ---> cf756be62977
Successfully built cf756be62977
Successfully tagged hello_world:latest
```

HELLO WORLD C++ STYLE - BUILD

```
~/cppnow18_docker_cpp/example/hello_world$ docker build -t hello_world .
Sending build context to Docker daemon  3.072kB
Step 1/5 : FROM ricejasonf/cppdock:linux_x64
 ---> 78b2d874d92c
Step 2/5 : COPY main.cpp /opt/build/
 ---> 2004574680e6
Step 3/5 : WORKDIR /opt/build
Removing intermediate container 6ac6b1d93c0b
 ---> e1fe223d84da
Step 4/5 : RUN clang++ -stdlib=libc++ -lc++abi main.cpp
 ---> Running in b5ccc39efc24
Removing intermediate container b5ccc39efc24
 ---> c6c593a12857
Step 5/5 : CMD ["./a.out"]
 ---> Running in 6c243b494d7f
Removing intermediate container 6c243b494d7f
 ---> cf756be62977
Successfully built cf756be62977
Successfully tagged hello_world:latest
```

HELLO WORLD C++ STYLE - BUILD

```
~/cppnow18_docker_cpp/example/hello_world$ docker build -t hello_world .
Sending build context to Docker daemon  3.072kB
Step 1/5 : FROM ricejasonf/cppdock:linux_x64
 ---> 78b2d874d92c
Step 2/5 : COPY main.cpp /opt/build/
 ---> 2004574680e6
Step 3/5 : WORKDIR /opt/build
Removing intermediate container 6ac6b1d93c0b
 ---> e1fe223d84da
Step 4/5 : RUN clang++ -stdlib=libc++ -lc++abi main.cpp
 ---> Running in b5ccc39efc24
Removing intermediate container b5ccc39efc24
 ---> c6c593a12857
Step 5/5 : CMD ["./a.out"]
 ---> Running in 6c243b494d7f
Removing intermediate container 6c243b494d7f
 ---> cf756be62977
Successfully built cf756be62977
Successfully tagged hello_world:latest
```

HELLO WORLD C++ STYLE – RUN

```
$ docker run hello_world
```

HELLO WORLD C++ STYLE – RUN

```
$ docker run hello_world
Hello, world!
```

MULTI-STAGE BUILDS

```
#include <iostream>

int main() {
    constexpr auto train = R"(
                                                                  _|"|_   _|"|_
  ___          _                  __   ___    _|"|_   _|"|_
 |   \    ___    __    |_|__   ___    _ _    /"/  / __|  |_   _| |_   _|
 | |) |  / _ \  / _|   | / /  / -_)   | '_|   / /  | (__    |_|    |_|
 |___/   \___/  \__|_  |_\_\  \___|  _|_|_  _/_/_  \___|  _____  _____
_|""""|_|""""|_|""""|_|""""|_|""""|_|""""|_|""""|_|""""|_|     |_|     |
"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'

)";

    std::cout << train;
}
```

MULTI-STAGE BUILDS

```
cmake_minimum_required(VERSION 3.9)
add_executable(train main.cpp)
add_definitions(-std=c++1z)
install(TARGETS train RUNTIME DESTINATION bin)
```

MULTI-STAGE BUILDS

```
FROM ricejasonf/cppdock:linux_x64 as linux

    COPY main.cpp /opt/src/
    COPY CMakeLists.txt /opt/src/
    WORKDIR /opt/build
    RUN cmake \
            -DCMAKE_TOOLCHAIN_FILE=/opt/toolchain.cmake \
            -DCMAKE_INSTALL_PREFIX=/opt/install \
            /opt/src \
     && cmake --build . --target train \
     && cmake --build . --target install

FROM ricejasonf/cppdock:emscripten as nodejs

    COPY main.cpp /opt/src/
    COPY CMakeLists.txt /opt/src/
    WORKDIR /opt/build
    RUN cmake \
            -DCMAKE_TOOLCHAIN_FILE=/opt/toolchain.cmake \
            -DCMAKE_INSTALL_PREFIX=/opt/install \
            /opt/src \
     && cmake --build . --target train \
     && cmake --build . --target install

FROM ubuntu:bionic

    COPY --from=linux  /opt/install/ /opt/install
    COPY --from=nodejs /opt/install/ /opt/install

    CMD cp -R /opt/install/* /opt/target/
```

MULTI-STAGE BUILDS

```
FROM ricejasonf/cppdock:linux_x64 as linux

    COPY main.cpp /opt/src/
    COPY CMakeLists.txt /opt/src/
    WORKDIR /opt/build
    RUN cmake \
            -DCMAKE_TOOLCHAIN_FILE=/opt/toolchain.cmake \
            -DCMAKE_INSTALL_PREFIX=/opt/install \
            /opt/src \
     && cmake --build . --target train \
     && cmake --build . --target install
```

```
FROM ricejasonf/
    COPY main.cp
    COPY CMakeL
    WORKDIR /op
    RUN cmake \
            -DC
            -DC
            /op
     && cmake -
     && cmake -
FROM ricejasonf/
    COPY main.cp
    COPY CMakeL
    WORKDIR /op
    RUN cmake \
            -DC
            -DC
            /opt/src \
     && cmake --build . --target train \
     && cmake --build . --target install

FROM ubuntu:bionic

    COPY --from=linux  /opt/install/ /opt/install
    COPY --from=nodejs /opt/install/ /opt/install

    CMD cp -R /opt/install/* /opt/target/
```

MULTI-STAGE BUILDS

```
FROM ricejasonf/cppdock:emscripten as nodejs

    COPY main.cpp /opt/src/
    COPY CMakeLists.txt /opt/src/
    WORKDIR /opt/build
    RUN cmake \
            -DCMAKE_TOOLCHAIN_FILE=/opt/toolchain.cmake \
            -DCMAKE_INSTALL_PREFIX=/opt/install \
            /opt/src \
     && cmake --build . --target train \
     && cmake --build . --target install
```

MULTI-STAGE BUILDS

```
FROM ricejasonf/cppdock:linux_x64 as linux

    COPY main.cpp /opt/src/
    COPY CMakeLists.txt /opt/src/
    WORKDIR /opt/build
    RUN cmake \
            -DCMAKE_TOOLCHAIN_FILE=/opt/toolchain.cmake \
            -DCMAKE_INSTALL_PREFIX=/opt/install \
            /opt/src \
     && cmake --build . --target train \
     && cmake --build . --target install

FROM ubuntu:bionic

    COPY --from=linux  /opt/install/ /opt/install
    COPY --from=nodejs /opt/install/ /opt/install

    CMD cp -R /opt/install/* /opt/target/
```

MULTI-STAGE BUILDS

```
$ docker build -t docker_train .

...
```

MULTI-STAGE BUILDS

```
$ docker run -v $(pwd)/target:/opt/target docker_train
```

MULTI-STAGE BUILDS

```
$ docker run -v $(pwd)/target:/opt/target docker_train
```

MULTI-STAGE BUILDS

```
$ docker run -v $(pwd)/target:/opt/target docker_train
```

MULTI-STAGE BUILDS

```
$ docker run -v $(pwd)/target:/opt/target docker_train
$
```

MULTI-STAGE BUILDS

```
$ docker run -v $(pwd)/target:/opt/target docker_train
$ node ./target/bin/train.js
```

MULTI-STAGE BUILDS

```
$ docker run -v $(pwd)/target:/opt/target docker_train
$ node ./target/bin/train.js
```

MULTI-STAGE BUILDS

```
$ docker run -v $(pwd)/target:/opt/target docker_train
$ node ./target/bin/train.js
```

BUILD A TOOLCHAIN

```
ARG EMSCRIPTEN_TAG=1.37.39
FROM ubuntu:bionic as fastcomp_build
  ARG EMSCRIPTEN_TAG

  RUN apt-get update && apt-get -y \
    install curl tar git cmake build-essential python

  RUN git clone --depth 1 \
                -b $EMSCRIPTEN_TAG \
                https://github.com/kripken/emscripten-fastcomp.git \
   && cd emscripten-fastcomp/tools/ \
   && git clone --depth 1 \
                -b $EMSCRIPTEN_TAG \
                https://github.com/kripken/emscripten-fastcomp-clang.git clang \
   && cd ../projects \
   && git clone --depth 1 https://github.com/llvm-mirror/libcxx.git \
   && git clone --depth 1 https://github.com/llvm-mirror/libcxxabi.git \
   && cd ../ && mkdir build && cd build \
   && cmake \
      -DCMAKE_BUILD_TYPE=Release \
      -DCMAKE_INSTALL_PREFIX=/usr/local \
      -DLLVM_TARGETS_TO_BUILD="X86;ARM;JSBackend" \
      -DLLVM_INCLUDE_EXAMPLES=OFF -DLLVM_INCLUDE_TESTS=OFF \
      -DCLANG_INCLUDE_EXAMPLES=OFF -DCLANG_INCLUDE_TESTS=OFF \
      .. \
   && make -j4 && make install

FROM ubuntu:bionic

  COPY --from=fastcomp_build /usr/local/lib /usr/local/lib
  COPY --from=fastcomp_build /usr/local/include /usr/local/include
  COPY --from=fastcomp_build /usr/local/bin /usr/local/bin
```

BUILD A TOOLCHAIN

```
ARG EMSCRIPTEN_TAG=1.37.39
FROM ubuntu:bionic as fastcomp_build
  ARG EMSCRIPTEN_TAG
```

BUILD A TOOLCHAIN

```
RUN apt-get update && apt-get -y \
  install curl tar git cmake build-essential python
```

BUILD A TOOLCHAIN

```
RUN git clone --depth 1 \
                -b $EMSCRIPTEN_TAG \
                https://github.com/kripken/emscripten-fastcomp.git \
    && cd emscripten-fastcomp/tools/ \
    && git clone --depth 1 \
                -b $EMSCRIPTEN_TAG \
                https://github.com/kripken/emscripten-fastcomp-clang.git \
                clang \
    && cd ../projects \
    && git clone --depth 1 https://github.com/llvm-mirror/libcxx.git \
    && git clone --depth 1 https://github.com/llvm-mirror/libcxxabi.git \
    && cd ../ && mkdir build && cd build \
```

BUILD A TOOLCHAIN

```
&& cmake \
    -DCMAKE_BUILD_TYPE=Release \
    -DCMAKE_INSTALL_PREFIX=/usr/local \
    -DLLVM_TARGETS_TO_BUILD="X86;ARM;JSBackend" \
    -DLLVM_INCLUDE_EXAMPLES=OFF -DLLVM_INCLUDE_TESTS=OFF \
    -DCLANG_INCLUDE_EXAMPLES=OFF -DCLANG_INCLUDE_TESTS=OFF\
    .. \
&& make -j4 && make install
```

BUILD A TOOLCHAIN

```
FROM ubuntu:bionic

  COPY --from=fastcomp_build /usr/local/lib /usr/local/lib
  COPY --from=fastcomp_build /usr/local/include /usr/local/include
  COPY --from=fastcomp_build /usr/local/bin /usr/local/bin
```

BUILD A TOOLCHAIN – SDK

```
ARG EMSCRIPTEN_TAG=1.37.39
FROM ricejasonf/emscripten_fastcomp:$EMSCRIPTEN_TAG
  ARG EMSCRIPTEN_TAG

  RUN apt-get update && apt-get -y install \
    python default-jre-headless curl tar xz-utils build-essential \
      cmake git python \
    && echo '. /usr/share/bash-completion/bash_completion && set -o vi' >> /root/.bashrc \
    && echo 'set hlsearch' >> /root/.vimrc

  WORKDIR /usr/local/src

  # node
  RUN curl -O https://nodejs.org/dist/v6.9.5/node-v6.9.5-linux-x64.tar.xz \
   && tar -xvf node-v6.9.5-linux-x64.tar.xz \
   && cp -r node-v6.9.5-linux-x64/* /usr/local/ \
   && rm -f node-v6.9.5-linux-x64.tar.xz \
   && rm -rf node-v6.9.5-linux-x64

  # Emscripten SDK
  RUN git clone --depth 1 -b $EMSCRIPTEN_TAG https://github.com/kripken/emscripten.git \
   && rm -rf emscripten/tests

  WORKDIR /usr/local/src/emscripten
  RUN ./emcc -v \
   && ./embuilder.py build ALL

  ENV CC=/usr/local/bin/clang \
      CXX=/usr/local/bin/clang++ \
      LD_LIBRARY_PATH=/usr/local/lib \
      EMCC_SKIP_SANITY_CHECK=1

  RUN echo 'export EMCC_SKIP_SANITY_CHECK=1' >> /root/.bashrc
```

BUILD A TOOLCHAIN

```
ARG EMSCRIPTEN_TAG=1.37.39
FROM ricejasonf/emscripten_fastcomp:$EMSCRIPTEN_TAG
  ARG EMSCRIPTEN_TAG
```

```
ARG EMSC
FROM ric
  ARG EM

  RUN ap
    pyth
      cm
    && e
    && e

  WORKDI

  # node
  RUN cu
   && ta
   && cp
   && rm
   && rm

  # Emsc
  RUN gi
   && rm

  WORKDI
  RUN ./
   && ./

  ENV CC
      CX
      LD
      EM

  RUN echo 'export EMCC_SKIP_SANITY_CHECK=1' >> /root/.bashrc
```

BUILD A TOOLCHAIN

```
RUN apt-get update && apt-get -y install \
      python default-jre-headless curl \
      tar xz-utils build-essential \
      cmake git python
```

```
ARG EMSC
FROM ric
  ARG EM

  RUN ap
    pyth
      cm
    && e
    && e

  WORKDI

  # node
  RUN cu
   && ta
   && cp
   && rm
   && rm

  # Emsc
  RUN gi
   && rm

  WORKDI
  RUN ./
   && ./

  ENV CC
      CX
      LD
      EM

  RUN echo 'export EMCC_SKIP_SANITY_CHECK=1' >> /root/.bashrc
```

BUILD A TOOLCHAIN

```
# node
RUN curl -O https://nodejs.org/dist/v6.9.5/node-v6.9.5-linux-x64.tar.xz \
 && tar -xvf node-v6.9.5-linux-x64.tar.xz \
 && cp -r node-v6.9.5-linux-x64/* /usr/local/ \
 && rm -f node-v6.9.5-linux-x64.tar.xz \
 && rm -rf node-v6.9.5-linux-x64
```

```
ARG EMSC
FROM ric
  ARG EM

  RUN ap
    pyth
      cr
    && e
    && e

  WORKDI

  # node
  RUN cu
   && ta
   && cp
   && rm
   && rm

  # Emsc
  RUN gi
   && rm

  WORKDI
  RUN ./
   && ./

  ENV CC
      CX
      LI
      EM

  RUN echo 'export EMCC_SKIP_SANITY_CHECK=1' >> /root/.bashrc
```

BUILD A TOOLCHAIN

```
# Emscripten SDK
RUN git clone --depth 1 -b $EMSCRIPTEN_TAG \
      https://github.com/kripken/emscripten.git \
  && rm -rf emscripten/tests
```

```
ARG EMSC
FROM ric
  ARG EM

  RUN ap
    pyth
      cr
    && e
    && e

  WORKDI

  # node
  RUN cu
   && ta
   && cp
   && rm
   && rm

  # Emsc
  RUN gi
   && rm

  WORKDI
  RUN ./
   && ./

  ENV CC
      CX
      LD
      EM

  RUN echo 'export EMCC_SKIP_SANITY_CHECK=1' >> /root/.bashrc
```

BUILD A TOOLCHAIN

```
WORKDIR /usr/local/src/emscripten
RUN ./emcc -v \
 && ./embuilder.py build ALL

ENV CC=/usr/local/bin/clang \
    CXX=/usr/local/bin/clang++ \
    LD_LIBRARY_PATH=/usr/local/lib \
    EMCC_SKIP_SANITY_CHECK=1
```

CONTRIBUTING TO AN OPEN SOURCE PROJECT

```
FROM ubuntu:bionic

RUN apt-get update && apt-get install -yq \
    gcc-7 build-essential cmake doxygen valgrind npm nodejs

RUN npm install -g http-server

WORKDIR /opt/build
CMD cmake /opt/src && /bin/bash
```

CONTRIBUTING TO AN OPEN SOURCE PROJECT

```
docker build --rm -t hana_build .
```

CONTRIBUTING TO AN OPEN SOURCE PROJECT

```
docker run --rm -it \
  -p 8080:8080 \
  -v $(pwd):/opt/src:ro \
  hana_build
```

CPPDOCK

CPPDOCK

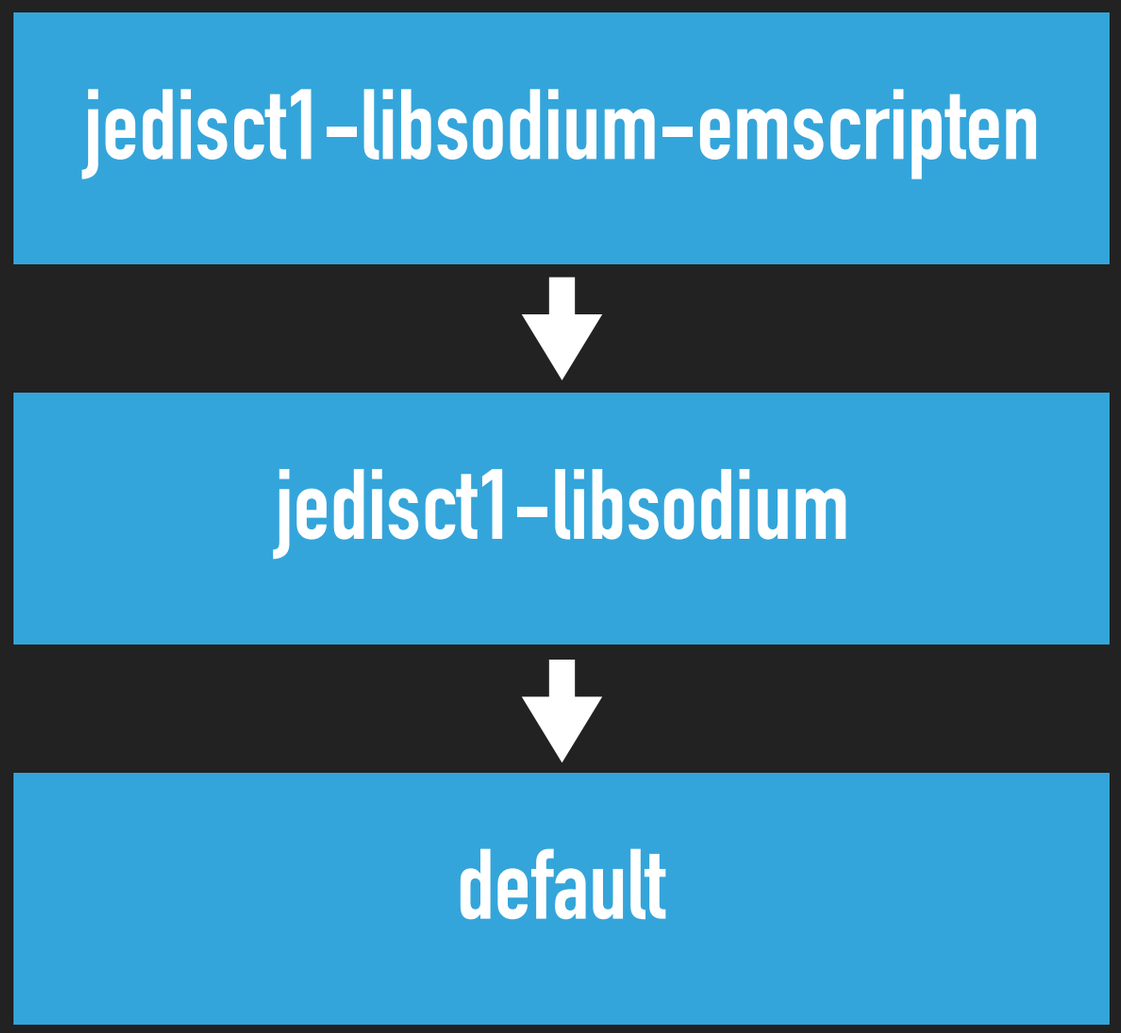
```
{
  "cppdock": {
    "name": "nbdl"
  },
  "platforms": {
    "develop": {
      "type": "linux_x64",
      "deps": [
        [
          {
            "name": "boostorg/callable_traits",
            "revision": "684dfbd7dfbdd0438ef3670be10002ca33a71715",
            "tag": "master"
          }
        ],
```

RECIPES (JUST SHELL SCRIPTS)

```
#!/bin/bash

mkdir build && cd build \
&& cmake \
    -DCMAKE_TOOLCHAIN_FILE=/opt/toolchain.cmake \
    -DCMAKE_INSTALL_PREFIX=/opt/install \
    -DCMAKE_BUILD_TYPE=Release \
    .. \
&& make install
```

RECIPE NAME RESOLUTION



RECIPES – LIBSODIUM / EMSCRIPTEN

```
#!/bin/bash

export PATH=$PATH:/usr/local/src/emscripten
apt-get update && apt-get install -yq libtool autoconf
chmod u+x ./autogen.sh
chmod u+x ./dist-build/emscripten.sh
./autogen.sh
./dist-build/emscripten.sh --standard
cp -r libsodium-js/* /opt/install/
```

CPPDOCK BUILD

```
$ cppdock build develop
```

CPPDOCK

```
$ cppdock dev develop
Sending build context to Docker daemon   6.03MB
Step 1/3 : FROM nbdl_build:develop
 ---> 85200aec045c
Step 2/3 : WORKDIR /opt/build
 ---> Using cache
 ---> 60685105a1bf
Step 3/3 : CMD cmake         -DCMAKE_BUILD_TYPE=Debug         -DCMAKE_TOOLCHAIN_FILE='/opt/toolchain.cmake'         /opt/src         && /bin/bash
 ---> Using cache
 ---> 3421d6c7170e
Successfully built 3421d6c7170e
Successfully tagged nbdl_dev:develop


    Finished building nbdl_dev:develop.

-- The C compiler identification is Clang 5.0.0
-- The CXX compiler identification is Clang 5.0.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Looking for pthread.h
-- Looking for pthread.h - found
-- Looking for pthread_create
-- Looking for pthread_create - not found
-- Looking for pthread_create in pthreads
-- Looking for pthread_create in pthreads - not found
-- Looking for pthread_create in pthread
-- Looking for pthread_create in pthread - found
-- Found Threads: TRUE
-- Configuring done
-- Generating done
-- Build files have been written to: /opt/build
root@62c68f22783a:/opt/build#
```

CPPDOCK

```
root@62c68f22783a:/opt/build# make check
```

CPPDOCK

```
$ cppdock dev develop
```

THANK YOU

- https://xkcd.com/1988/
- https://docs.docker.com/v17.09/engine/userguide/
- http://mr.gy/blog/build-vm-image-with-docker.html
- https://github.com/ricejasonf/cppnow18_docker_cpp
- https://github.com/ricejasonf/cppdock