Docker  C++

Adrian Ostrowski

https://github.com/aostrowski | @adr_ostrowski

http://aostrowski.github.io

Piotr Gaczkowski

https://github.com/DoomHammer | @doomhammerng

https://doomhammer.info

Agenda

- Docker refresher: consuming and producing packages
- CMake: building, installing and packaging C++
- Creating Docker images using CMake deliverables
- Multi-stage builds
- Debugging C++ code inside containers
- Managing toolchains with Docker
- Running external services
- Docker Compose
- CI/CD with Docker


docker

Running Docker containers

```
docker run -i -t --rm \
    -e FOO=bar \
    -p 8080:80/udp \
    -v /path/on/host:/path/in/container:ro \
    image:latest
```

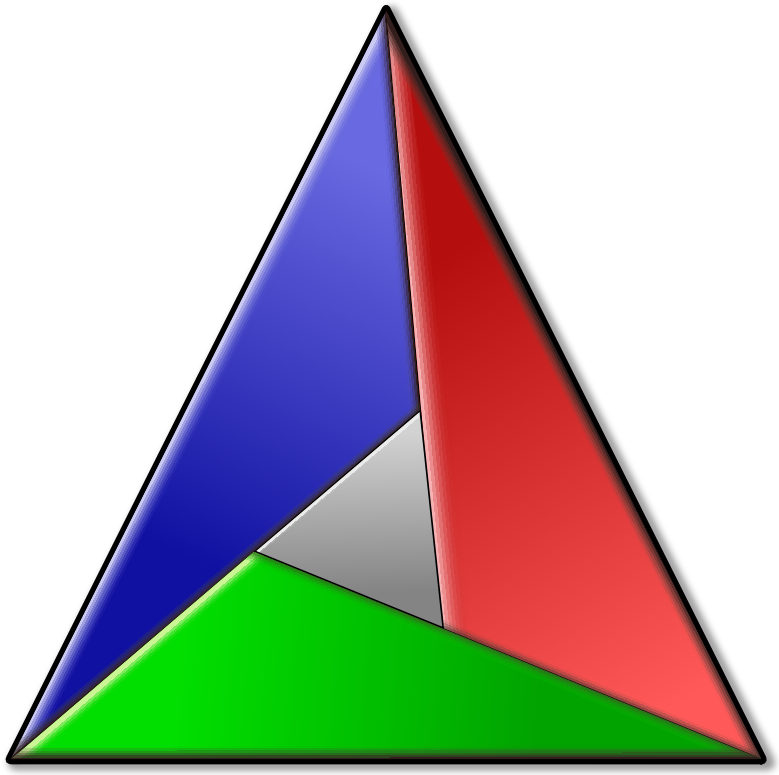
Building Docker containers

```
docker build . -t image:latest
```

Dockerfile

```
FROM ubuntu:rolling
ADD files_from_host path_in_container
RUN apt-get update && \
    apt-get -y install my_package && \
    apt-get autoremove -y && \
    apt-get clean && \
    rm -r /var/lib/apt/lists/*
ENTRYPOINT ["my_command"]
EXPOSE 8080
```

Building C++ code



CMake: simple target

```
add_executable(customer main.cpp)
target_compile_features(customer PRIVATE cxx_std_17)
target_link_libraries(customer PRIVATE libcustomer)
set_target_properties(customer PROPERTIES CXX_EXTENSIONS OFF)
target_compile_options(customer PRIVATE ${MY_COMPILE_FLAGS})
```

CMake: installation

```
install(TARGETS libcustomer customer)

# optionally:
#  - headers
#  - licenses
#  - Config and Targets files
```

CPack: creating packages

```
set(CPACK_PACKAGE_VENDOR "Authors")
set(CPACK_PACKAGE_CONTACT "author@example.com")
set(CPACK_PACKAGE_DESCRIPTION_SUMMARY
    "Library and app for the Customer microservice")
set(CPACK_PACKAGE_VERSION_MAJOR ${PROJECT_VERSION_MAJOR})
set(CPACK_PACKAGE_VERSION_MINOR ${PROJECT_VERSION_MINOR})
set(CPACK_PACKAGE_VERSION_PATCH ${PROJECT_VERSION_PATCH})
```

CPack: adding a DEB package

```
list(APPEND CPACK_GENERATOR DEB)
set(CPACK_DEBIAN_PACKAGE_DEPENDS "${CPACK_DEBIAN_PACKAGE_DEPENDS}
    libcpprest2.10 (>= 2.10.2-6)")
```

CPack: finishing touches

```
set(CPACK_SOURCE_IGNORE_FILES /.git /dist /.*build.* /\\\\.DS_Store)

include(CPack)
```

Containerizing: Dockerfile

```
FROM ubuntu:rolling
ADD Customer-1.0.0-Linux.deb .
RUN apt-get update && \
    apt-get -y --no-install-recommends install \
        ./Customer-1.0.0-Linux.deb && \
    apt-get autoremove -y && \
    apt-get clean && \
    rm -r /var/lib/apt/lists/* Customer-1.0.0-Linux.deb
ENTRYPOINT ["/usr/bin/customer"]
EXPOSE 8080
```

CMake: Finding Docker

```
find_program(Docker_EXECUTABLE docker)
if (NOT Docker_EXECUTABLE)
    message(FATAL_ERROR "Docker not found")
endif()
```

CMake: target for packaging

```
add_custom_target(
    customer-deb
    COMMENT "Creating Customer DEB package"
    COMMAND ${CMAKE_CPACK_COMMAND} -G DEB
    WORKING_DIRECTORY ${PROJECT_BINARY_DIR}
    VERBATIM)
add_dependencies(customer-deb customer)
```

CMake: Docker image target

```
configure_file(${PROJECT_SOURCE_DIR}/Dockerfile
               ${PROJECT_BINARY_DIR}/Dockerfile COPYONLY)

add_custom_target(
    docker
    COMMENT "Preparing Docker image"
    COMMAND ${Docker_EXECUTABLE} build ${PROJECT_BINARY_DIR} -t
            customer:${PROJECT_VERSION} -t customer:latest
    VERBATIM)
add_dependencies(docker customer-deb)
```

Achieving reproducible builds

- we have a container

Achieving reproducible builds

- we have a container...
- but built using developer's environment

Achieving reproducible builds

- we have a container...
- but built using developer's environment
- let's contain the whole build instead

Multiphase builds

- Docker feature for complex builds

Multiphase builds

- Docker feature for complex builds
- resulting in minimal-sized images

Multiphase builds

- Docker feature for complex builds
- resulting in minimal-sized images
- can use multiple FROM statements

Multiphase builds

- Docker feature for complex builds
- resulting in minimal-sized images
- can use multiple FROM statements
- we can have one step for building

Multiphase builds

- Docker feature for complex builds
- resulting in minimal-sized images
- can use multiple FROM statements
- we can have one step for building
- and another for gathering results

Multiphase Dockerfile

```
FROM builder:latest AS build
WORKDIR /build_dir
RUN cmake --build . --target customer-deb

FROM ubuntu:rolling
COPY --from=build /build_dir/Customer-1.0.0-Linux.deb .
RUN apt-get update && \
    apt-get -y --no-install-recommends install \
        ./Customer-1.0.0-Linux.deb && \
    apt-get autoremove -y && apt-get clean && \
    rm -r /var/lib/apt/lists/* Customer-1.0.0-Linux.deb
ENTRYPOINT ["/usr/bin/customer"]
EXPOSE 8080
```

Debugging

- remote, using gdbserver

Debugging

- remote, using gdbserver

```
FROM customer:latest
RUN apt-get update && apt-get install gdbserver
```

Managing toolchains with Docker

How do you make sure everybody is running the same toolchain?

Managing toolchains with Docker

How do you make sure everybody is running the same toolchain?

- Pre-installed machine from IT

Managing toolchains with Docker

How do you make sure everybody is running the same toolchain?

- Pre-installed machine from IT
- VM images

Managing toolchains with Docker

How do you make sure everybody is running the same toolchain?

- Pre-installed machine from IT
- VM images
- chroot/container images

Managing toolchains with Docker

How do you make sure everybody is running the same toolchain?

- Pre-installed machine from IT
- VM images
- chroot/container images
- application images

Managing toolchains with Docker

How do you make sure everybody is running the same toolchain?

- Pre-installed machine from IT
- VM images
- chroot/container images
- application images

What else?

Managing toolchains with Docker

How do you make sure everybody is running the same toolchain?

- Pre-installed machine from IT
- VM images
- chroot/container images
- application images

What else?

And then: how do you upgrade this snowflake solution?

Conan

Are you familiar with Conan?

Conan

Are you familiar with Conan?

Conan solves a lot of problems related to C++ dependency management.

Conan

Are you familiar with Conan?

Conan solves a lot of problems related to C++ dependency management.

If you know cpan, pip, npm, or gem, Conan is kinda similar.

Conan

Are you familiar with Conan?

Conan solves a lot of problems related to C++ dependency management.

If you know cpan, pip, npm, or gem, Conan is kinda similar.

But since it deals with C++, it's a **bit** more complicated.

Conan

Are you familiar with Conan?

Conan solves a lot of problems related to C++ dependency management.

If you know cpan, pip, npm, or gem, Conan is kinda similar.

But since it deals with C++, it's a **bit** more complicated.

Main strength: install C++ libraries without superuser permissions.

Conan base images

Conan provides nice base images on Docker Hub

Different versions of GCC, from GCC 4.6 up to GCC 11

Different versions of Clang, from 3.8 to 10

Visual Studio 14 and 15 images (not redistributable)

Conan base images

x86_64, armv7, armv7hf, armv8, Android toolchains

May be used as JNLP slaves for use with Jenkins

Possible to use with Travis and other container-based CI/CD systems

Only suitable for linux/amd64 hosts (unless Visual Studio containers)

Building with Docker

```
docker run --rm -ti -v conan-gcc10:/home/conan/.conan \
-v $PWD:/home/conan/src -w /home/conan/src/dockerbuild \
conanio/gcc10 conan install ..

docker run --rm -ti -v conan-gcc10:/home/conan/.conan \
-v $PWD:/home/conan/src -w /home/conan/src/dockerbuild \
conanio/gcc10 cmake ..

docker run --rm -ti -v conan-gcc10:/home/conan/.conan \
-v $PWD:/home/conan/src -w /home/conan/src/dockerbuild \
conanio/gcc10 cmake --build .
```

Building with Docker

Or:

```
docker run --rm -ti -v conan-gcc10:/home/conan/.conan \
-v $PWD:/home/conan/src -w /home/conan/src/dockerbuild \
conanio/gcc10 \
bash -c 'conan install .. && cmake .. && cmake --build .'
```

Building with Docker

Also:

```
alias conaniogcc10='docker run --rm -ti -v conan-gcc10:/home/conan/.con
conaniogcc10 conan install ..
conaniogcc10 cmake ..
conaniogcc10 cmake --build .
```

Upgrading the toolchain

Upgrade is as easy as switching `gcc10` to `gcc11` in the command line. That's it.

```
docker run --rm -ti -v conan-gcc11:/home/conan/.conan \
-v $PWD:/home/conan/src -w /home/conan/src/dockerbuild \
conanio/gcc11 \
bash -c 'conan install .. && cmake .. && cmake --build .'
```

What's inside

A compiler with its toolchain (duh!)

wget and curl

git and subversion

nasm

make and ninja

pkg-config

autotools

JFrog CLI

Conan

CMake (3.18.2)

Python (Scons, Waf)

Android NDK (where suitable)

Conan profiles for cross-compilers (where suitable)

Anything lacking? No problem!

```
FROM conanio/gcc10
RUN sudo apt-get update && \
    sudo apt-get -y --no-install-recommends install qt5-default && \
    sudo apt-get autoremove -y && \
    sudo apt-get clean && \
    sudo rm -r /var/lib/apt/lists/*
```

Running external services

```
docker run --rm postgres

docker run --rm -e POSTGRES_PASSWORD=iLoveDocker postgres
```

Running external services

Also suitable for:

- Nginx
- Redis
- Other caches
- Other DBs
- IceCC scheduler
- Most of whatever else you need

Docker Compose

`docker-compose up` to run **anything** and **everything**

- Lets you automate container configuration
- Can set up and tear down other resources (networks and volumes) and manages their lifecycle
- Multiple levels of overrides possible
- Makes it easy to define and connect complex services

```yaml
version: "3.8"
services:
  redis:
    image: redis
    networks:
      - backend
  db:
    image: postgres
    volumes:
      - "postgres:/var/lib/postgresql/data"
    networks:
      - backend
  customer:
    image: customer:1.0.1
    networks:
      - backend
      - frontend
```

Building with Docker Compose

Besides running several services in parallel, Docker Compose can be used for documenting one-off commands.

Building with Docker Compose

Besides running several services in parallel, Docker Compose can be used for documenting one-off commands.

Use cases:

Building with Docker Compose

Besides running several services in parallel, Docker Compose can be used for documenting one-off commands.

Use cases:

- Multiple compilers side by side

Building with Docker Compose

Besides running several services in parallel, Docker Compose can be used for documenting one-off commands.

Use cases:

- Multiple compilers side by side
- Builds for different distros

Building with Docker Compose

Besides running several services in parallel, Docker Compose can be used for documenting one-off commands.

Use cases:

- Multiple compilers side by side
- Builds for different distros
- Easy cross-compilation

Building with Docker Compose

Besides running several services in parallel, Docker Compose can be used for documenting one-off commands.

Use cases:

- Multiple compilers side by side
- Builds for different distros
- Easy cross-compilation
- Compiling against different versions of dependencies

```yaml
version: "3.8"
services:
  gcc-10:
    image: conanio/gcc10
    volumes:
      - conan-gcc10:/home/conan/.conan
      - .:/home/conan/src
    working_dir: /home/conan/src/dockerbuild
  gcc-11:
    image: conanio/gcc11
    volumes:
      - conan-gcc11:/home/conan/.conan
      - .:/home/conan/src
    working_dir: /home/conan/src/dockerbuild
volumes:
  conan-gcc10:
  conan-gcc11:
```

Building with Docker Compose

```
docker-compose run gcc-10 \
bash -c 'conan install .. && cmake .. && cmake --build .'
```

CI/CD with Docker

CI/CD with Docker

- Jenkins

CI/CD with Docker

- Jenkins
- Travis

CI/CD with Docker

- Jenkins
- Travis
- CircleCI

CI/CD with Docker

- Jenkins
- Travis
- CircleCI
- Gitlab CI

CI/CD with Docker

- Jenkins
- Travis
- CircleCI
- Gitlab CI
- GitHub Actions

Gitlab CI

```
.build-ubuntu-lts:
    image: conanio/conangcc8
    script:
      - conan install ..
      - cmake ..
      - cmake --build .
    artifacts:
      paths:
        - build/coverage/
      expire_in: 1 week

[...]
```

Gitlab CI

```
[...]
build-gcc10:
    extends: .build-ubuntu-lts
    image: conanio/gcc10
    variables:
        CC: '/usr/bin/gcc-10'
        CXX: '/usr/bin/g++-10'
```

CircleCI

```
version: 2.1

executors:
  gcc10:
    docker:
      - image: conanio/gcc10
jobs:
  build:
    executor: gcc10
    steps:
      - run:
          command: conan install .. && cmake .. && cmake --build .
```

GitHub Actions

```yaml
jobs:
  build:
    strategy:
      matrix:
        docker_image:
          - conanio/gcc10

    steps:
      - uses: actions/checkout@master
      - uses: actions/setup-python@master
        with:
          python-version: '3.7'
      - name: Building and publish the package
        run: conan install .. && cmake .. && cmake --build .
```

Benefits of Docker for building

- Same toolchain and dependencies everywhere
- Easy to change the configuration (via a Dockerfile)
- Same toolchain locally as on CI/CD
- Easy to test new features
- Massive parallel builds with different settings manageable from a single file
- Easy to distribute the images
- Lower overhead than VMs

Benefits of Docker for building

- Power to the people

Story Time!

Hungry for more?

Buy our new best-selling book!

Featuring:

- More on building and packaging
- Designing quality software
- Leveraging C++20 features
- Microservices and cloud-native C++

Available on Packt and Amazon

Questions?

Thank you!

https://github.com/aostrowski | @adr_ostrowski

http://aostrowski.github.io

Thank you!

https://github.com/DoomHammer | @doomhammerng

https://doomhammer.info

Attributions

- Logo of Docker, a Linux container engine byi dotCloud, Inc., source
- Logo of Cmake, Cmake team. The original uploader was Francesco Betti Sorbelli at Italian Wikipedia.. Vectorized by Magasjukur2, source