# Dockerising a 32 bit binary

## Multi-Stage Dockerfile
## 32 bit app in a 64-bit Docker Container

**Maurice Castro**

# Dockerising a 32 bit binary

## A little more than building a basic container

Why?

3 techniques:

- Multi-Stage Dockerfile
- i386 architecture
- Using CMD and ENTRYPOINT

# Why

## The problem

- 32 bit is going from Ubuntu …
  - at very best we are going to have to jump through hoops
  - how do we keep our applications using ancient libraries going?
- How do we keep our Docker containers small
  - We really do not want to include the build environment in the container
- How do we make our command line docker (slightly) easier to use …
  - separate the command from the arguments

# Multi-Stage Dockerfile

## Compile in one docker run in another

**Dockerfile**

```
FROM i386/ubuntu:16.04 AS builder
RUN apt-get update -y && apt-get install -y make g++
WORKDIR /app
COPY . /app
RUN make dockerinstall

FROM ubuntu:16.04
RUN dpkg --add-architecture i386
RUN apt-get update -y
RUN apt-get install -y libc6-dbg libc6-dbg:i386 lib32stdc++6
COPY --from=builder /app/lib/lib*.so /usr/lib
RUN  ldconfig
COPY --from=builder /app/example  /usr/bin
CMD ["-h"]
ENTRYPOINT [“/usr/bin/example”]
```

**Makefile**

```
all: example

INC=inc
LIB=lib

example.o: example.cpp
	g++ -m32 -std=c++0x -c example.cpp -I$(INC)

example: example.o
	g++ -m32 -o example example.o -L$(LIB) -lancientlib

clean:
	rm -f example
	rm -f example.o

test: example
	env LD_LIBRARY_PATH=$(LIB) ./example 192.168.168.133

dockerinstall: example
	cp example /usr/bin
	cp $(LIB)/ancientlib.so /usr/local/lib
	cp $(LIB)/ancientlib.so.2.40.0 /usr/local/lib
```

# Multi-Stage Dockerfile

## First Stage

- In this case we get a 386 container and install our build tools
- This container is internally called “builder”
- Copy our code into the app directory
- Use make to build it and install to locations we want
  - libraries are copied to /usr/local/lib & program (example) to /usr/bin

```
FROM i386/ubuntu:16.04 AS builder
RUN apt-get update -y && apt-get install -y make g++
WORKDIR /app
COPY . /app
RUN make dockerinstall
```

# Multi-Stage Dockerfile

## Second Stage

- In this case we get a standard container and customise it to run i386 binaries
- Copy our compiled code and libraries from “builder”
- Build the shared library cache

```
FROM ubuntu:16.04
RUN dpkg --add-architecture i386
RUN apt-get update -y
RUN apt-get install -y libc6-dbg libc6-dbg:i386 lib32stdc++6
COPY --from=builder /app/lib/lib*.so /usr/lib
RUN  ldconfig
COPY --from=builder /app/example  /usr/bin
CMD ["-h"]
ENTRYPOINT [“/usr/bin/example”]
```

- To build: docker build -t example .

# Multi-Stage Dockerfile

## What we got

- A container with only what is need to run our program
- A 64 bit container with a 32-bit program in it that runs

# i386 Architecture

## Two ways

- An ubuntu 16.04 i386 container

```
FROM i386/ubuntu:16.04 AS builder
```

- A 64 bit container with i386 architecture support

```
FROM ubuntu:16.04
RUN dpkg --add-architecture i386
```

# Using ENTRYPOINT & CMD

## It runs sort of like a standard command line program

- Using ENTRYPOINT & CMD together
  - ENTRYPOINT names the program to run
  - CMD defines the default arguments to the program
- In this example we display the help if no arguments are provided by the user of the container

```
CMD ["-h"]
ENTRYPOINT [“/usr/bin/example”]
```

- Display help: docker run example
- Pass an argument: docker run example 192.168.168.133

# References

## Sources


- Multi-Stage Dockerfile for building i386
  https://capstonec.com/2019/05/24/32bit-apps-in-a-64bit-docker-container/

- Using CMD and ENTRYPOINT
  https://dev.to/lasatadevi/docker-cmd-vs-entrypoint-34e0