

Debugging in Docker



Current Debugging Workflow

1. Write code
2. Compile
3. Run in GDB/Valgrind

Current Debugging Workflow

1. Write code
2. Compile
3. Run in GDB/Valgrind

Issues?

- Different version of g++/valgrind from CAEN
- Can't use Valgrind/GDB if you're on mac/windows

Alternatives?

- Go to a CAEN computer
- SSH into CAEN
- Virtual Machine
- **Docker!**

What is Docker?

- A form of virtualisation
- Containerise everything!

But That's Just a VM?

- It doesn't create the entire OS
- All images used shared resources

But That's Just a VM?

- It doesn't create the entire OS
- All images used shared resources

Which is Better?

Docker

- Is much quicker to start
- *Hella fast*
- Open source :D
- Almost **0** overhead

VM's

- Full isolation
- You know how it works

Getting Started

- [Download Docker](#) (if you haven't already)
- Start the Docker client
- `docker pull alpine`
- `docker run alpine`

Getting Started

- [Download Docker](#) (if you haven't already)
- Start the Docker client
- `docker pull alpine`
- `docker run alpine`
- Why didn't that do anything?
 - You didn't give it any commands
 - `docker run -it alpine` to open an interactive shell

What We Need

- A linux image ([Ubuntu 17.10](#))
- Valgrind, GDB, etc.

What We Need

- A linux image ([Ubuntu 17.10](#))
- Valgrind, GDB, etc.

Setting up an Image

- `docker pull ubuntu:artful`
- `docker run -it alpine`
- `apt-get update`
- `apt-get install g++ valgrind make`

Building the Docker Way

- Use a Dockerfile
 - This contains all the commands a user would call on the command line to assemble an image
 - Makes it easy to share images, since you only need to share a simple text file

```
# Build a simple ubuntu image with vim installed
FROM ubuntu:artful
RUN apt-get update
RUN apt-get install -y vim
CMD ["bash"]
```

```
$ docker build .
```

Accessing Your Files

We hate two options:

- Copy over files
 - `ADD . /DOCKER/PATH`
- Mount directory
 - `docker run -it -v "$(pwd):/DOCKER/PATH`

Time to Build a Debugging Container

The Dockerfile

Objective

- Build an image running ubuntu that has all the tools we need

The Dockerfile

Get the linux distro we want

```
# Using Ubuntu 17:01 (artful)  
FROM ubuntu:artful
```


The Dockerfile

Install everything we need

```
RUN apt-get update
RUN apt-get install -y g++
RUN apt-get install -y gcc
RUN apt-get install -y make
RUN apt-get install -y valgrind
RUN apt-get install -y vim
```

The Dockerfile

Get it ready to be run

```
# Set starting directory to /prog  
RUN mkdir /prog  
WORKDIR /prog
```

```
# Run bash  
CMD ["bash"]
```

The Dockerfile

```
# Using Ubuntu 17.10
FROM ubuntu:artful

RUN apt-get update
RUN apt-get install -y g++
RUN apt-get install -y gcc
RUN apt-get install -y make
RUN apt-get install -y valgrind
RUN apt-get install -y vim

# Set starting directory to home
RUN mkdir /prog
WORKDIR /prog

CMD ["bash"]
```

Build It

- If you're in the same directory as the Dockerfile

```
docker build -t docker-debugger .
```

- If not

```
docker build -f /PATH/TO/Dockerfile -t docker-debugger
```

Running It

- Run the most recent 'docker-debugger' image

```
# Run it
# Mount the current directory to /prog
docker run -it --rm --privileged \
  -v "$(pwd):/prog" docker-debugger:latest
```

Running It

What are all those other flags??

- `-it`: open an interactive shell
- `--rm`: remove the container on exit
- `--privileged`: give it permissions required for gdb/valgrind
- `-v "LOCAL_PATH:CONTAINER_PATH"`: Mount `LOCAL_PATH` on your container at `CONTAINER_PATH`

Is That It?

- This is a really simple use case
- Deploy a website anywhere
 - Have loads of images + a load balancer!

Is That It?

Container Orchestration

- Managing loads and loads of running containers
- Scaling as appropriate by adding or removing containers
- Distributing load between the containers
- Launching new containers on different machines if something fails