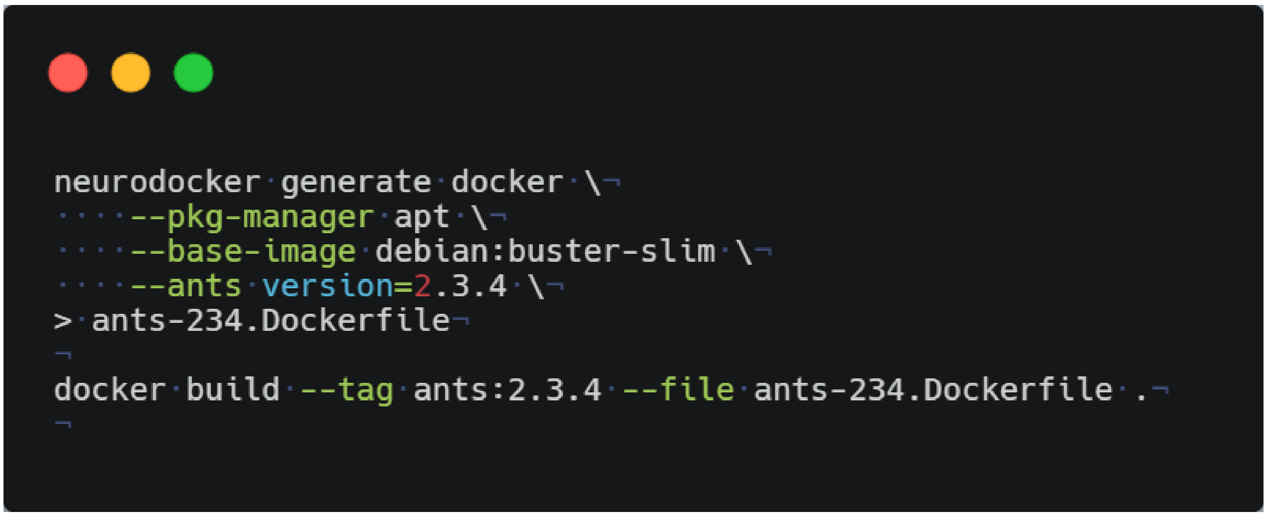
Running NeuroDesk on ARM

The project’s main goal was to test and run the NeuroDesk suite of medical imaging containers on ARM. For the purpose of testing, Advanced Normalization Tools (ANTs) was chosen. ANTs computes high-dimensional mappings to capture the statistics of brain structure and function [39].

Approach 1: Building the ANTs Docker image for ARM

The first approach involved trying to get ANTs running on ARM by using Neurodocker [40] to generate a Docker build file on ARM and building the image.

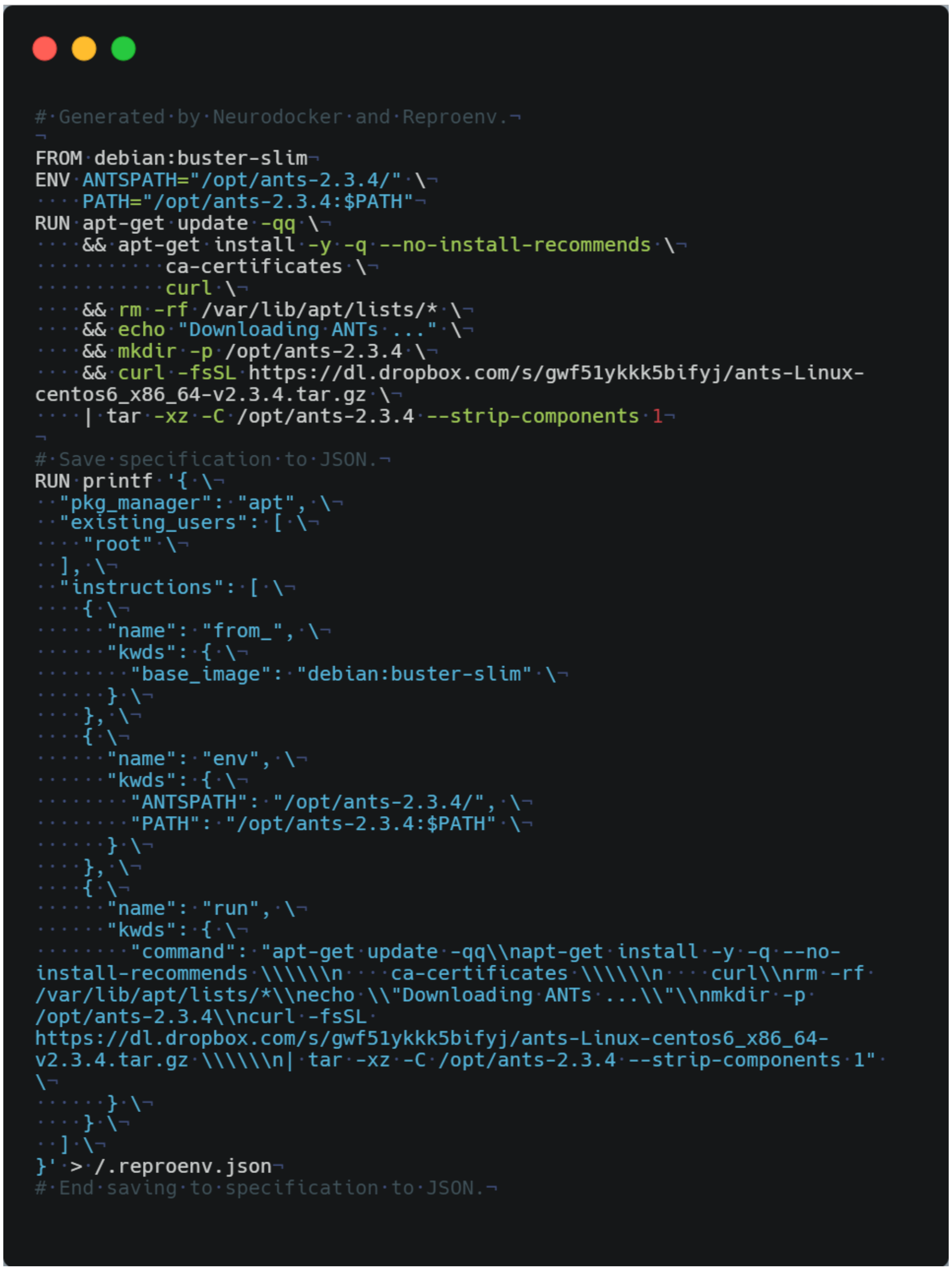


```
neurodocker generate docker \
    --pkg-manager apt \
    --base-image debian:buster-slim \
    --ants version=2.3.4 \
> ants-234.Dockerfile

docker build --tag ants:2.3.4 --file ants-234.Dockerfile .

```

Figure 8 Generating a Dockerfile for ANTs and building the image.



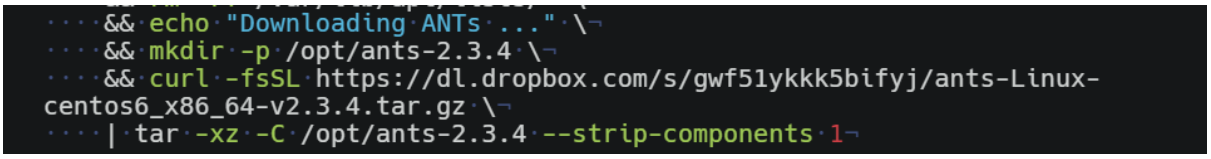
```
# Generated by Neurodocker and Reproenv.

FROM debian:buster-slim
ENV ANTSPATH="/opt/ants-2.3.4/" \
    PATH="/opt/ants-2.3.4:$PATH"
RUN apt-get update -qq \
    && apt-get install -y -q --no-install-recommends \
           ca-certificates \
           curl \
    && rm -rf /var/lib/apt/lists/* \
    && echo "Downloading ANTs ..." \
    && mkdir -p /opt/ants-2.3.4 \
    && curl -fsSL https://dl.dropbox.com/s/gwf51ykkk5bifyj/ants-Linux-
centos6_x86_64-v2.3.4.tar.gz \
    | tar -xz -C /opt/ants-2.3.4 --strip-components 1

# Save specification to JSON.
RUN printf '{ \
  "pkg_manager": "apt", \
  "existing_users": [ \
    "root" \
  ], \
  "instructions": [ \
    { \
      "name": "from_", \
      "kwds": { \
        "base_image": "debian:buster-slim" \
      } \
    }, \
    { \
      "name": "env", \
      "kwds": { \
        "ANTSPATH": "/opt/ants-2.3.4/", \
        "PATH": "/opt/ants-2.3.4:$PATH" \
      } \
    }, \
    { \
      "name": "run", \
      "kwds": { \
        "command": "apt-get update -qq\\napt-get install -y -q --no-
install-recommends \\\\\\n    ca-certificates \\\\\\n    curl\\nrm -rf 
/var/lib/apt/lists/*\\necho \\"Downloading ANTs ...\\"\\nmkdir -p 
/opt/ants-2.3.4\\ncurl -fsSL 
https://dl.dropbox.com/s/gwf51ykkk5bifyj/ants-Linux-centos6_x86_64-
v2.3.4.tar.gz \\\\\\n| tar -xz -C /opt/ants-2.3.4 --strip-components 1" 
\
      } \
    } \
  ] \
}' > /.reproenv.json
# End saving to specification to JSON.
```

Figure 9 The Neurodocker generated Dockerfile for ANTS.

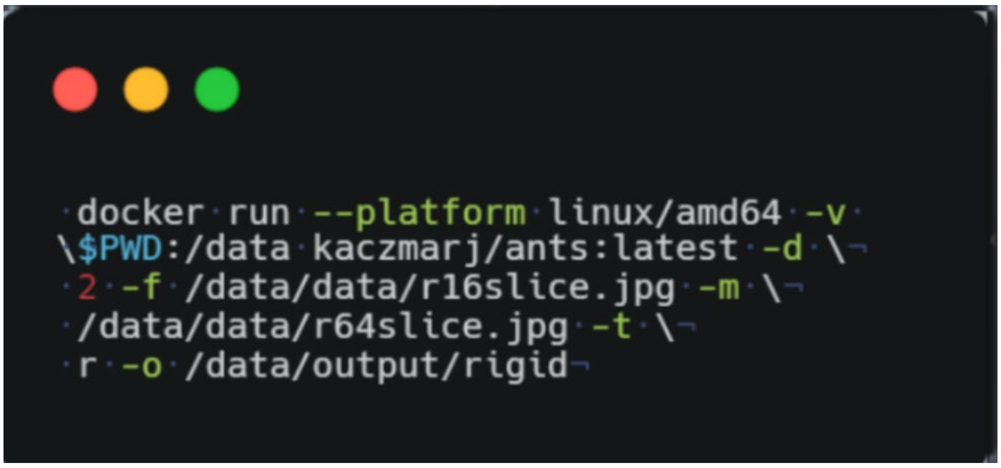
The image itself builds successfully, but unfortunately, during the build process, the Dockerfile pulls in an archive of pre-compiled x86_64 ANTs executables which are hosted on [Dropbox](). This meant that the executables would not run on ARM64.



```
    && echo "Downloading ANTs ..." \
    && mkdir -p /opt/ants-2.3.4 \
    && curl -fsSL https://dl.dropbox.com/s/gwf51ykkk5bifyj/ants-Linux-
centos6_x86_64-v2.3.4.tar.gz \
    | tar -xz -C /opt/ants-2.3.4 --strip-components 1
```

Figure 10 The Dockerfile pulls in precompiled x86 binaries

The next attempt was to try and use Docker's built-in architecture platform emulation mechanism[33].



```
docker run --platform linux/amd64 -v
\$PWD:/data kaczmarj/ants:latest -d \
2 -f /data/data/r16slice.jpg -m \
/data/data/r64slice.jpg -t \
r -o /data/output/rigid
```

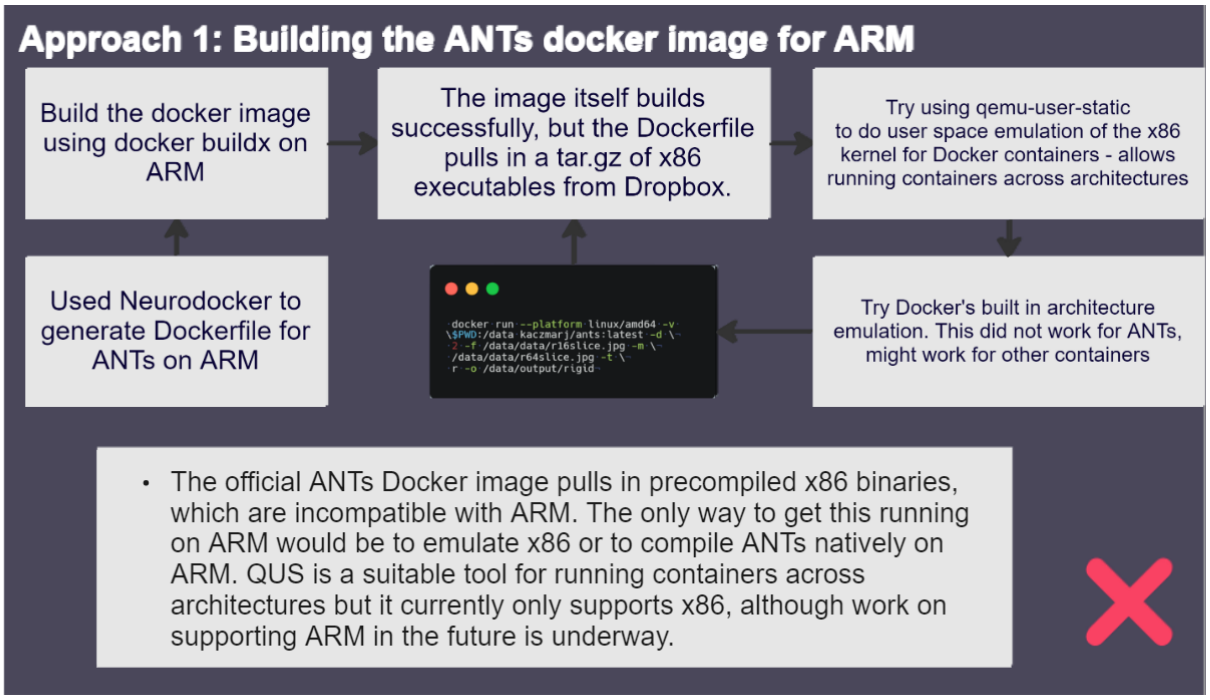
Figure 11 Using Docker's platform emulation feature

This did not work for ANTs, as the Docker image depends on the x86_64 executables being natively supported by the architecture. Still, it might work for other containers which do not pull in precompiled binaries at build time.

Finally, qemu-user-static [41] was used to do user-space emulation of the x86_64 architecture on ARM for Docker containers. **qemu-user-static** enables the execution of different multi-architecture containers by using QEMU and binfmt_misc. The Linux kernel has a feature called binfmt_misc (Miscellaneous Binary Format) that enables the recognition and passing of any executable file format to certain user space programs like emulators and virtual machines. It is one of several binary format handlers the kernel uses to get a user-space program ready to run [31].

This allows running containers across architectures. But on further probing, it was discovered that qemu-user-static currently only supported x86_64, although ARM support is under development. Once it has been ported to ARM, QUS promises to be a quick and easy method to run containers across architectures.

Summary for Approach 1: The official ANTs Docker Image pulls in precompiled x86_64 binaries, which are incompatible with ARM64. The only way to get this running on ARM64 would be to emulate the x86_64 architecture or compile ANTs natively on ARM64, which is precisely what the next two approaches will focus on. QUS is suitable for running containers across architectures, but it currently only supports x86_64, although work on supporting ARM64 in the future is underway.



Approach 1: Building the ANTs docker image for ARM

Build the docker image using docker buildx on ARM

The image itself builds successfully, but the Dockerfile pulls in a tar.gz of x86 executables from Dropbox.

Try using qemu-user-static to do user space emulation of the x86 kernel for Docker containers - allows running containers across architectures

Used Neurodocker to generate Dockerfile for ANTs on ARM

```
docker run --platform linux/amd64 -v
\$PWD:/data kaczmarj/ants:latest -d \
2 -f /data/data/r16slice.jpg -m \
/data/data/r64slice.jpg -t \
r -o /data/output/rigid
```

Try Docker's built in architecture emulation. This did not work for ANTs, might work for other containers

- The official ANTs Docker image pulls in precompiled x86 binaries, which are incompatible with ARM. The only way to get this running on ARM would be to emulate x86 or to compile ANTs natively on ARM. QUS is a suitable tool for running containers across architectures but it currently only supports x86, although work on supporting ARM in the future is underway.

Figure 12 Approach 1: Building the ANTs docker image for ARM

Approach 2: Emulating the x86_64 architecture in QEMU

The following approach involved emulating the whole x86_64 compute architecture on ARM64 using QEMU.

QEMU is a free and open-source emulator (Quick EMUlator). Dynamic binary translation is used to simulate the machine's processor, and various hardware and device models are provided for the machine to run different guest operating systems. To run virtual machines at nearly native performance, it can integrate with Kernel-based Virtual Machine (KVM).

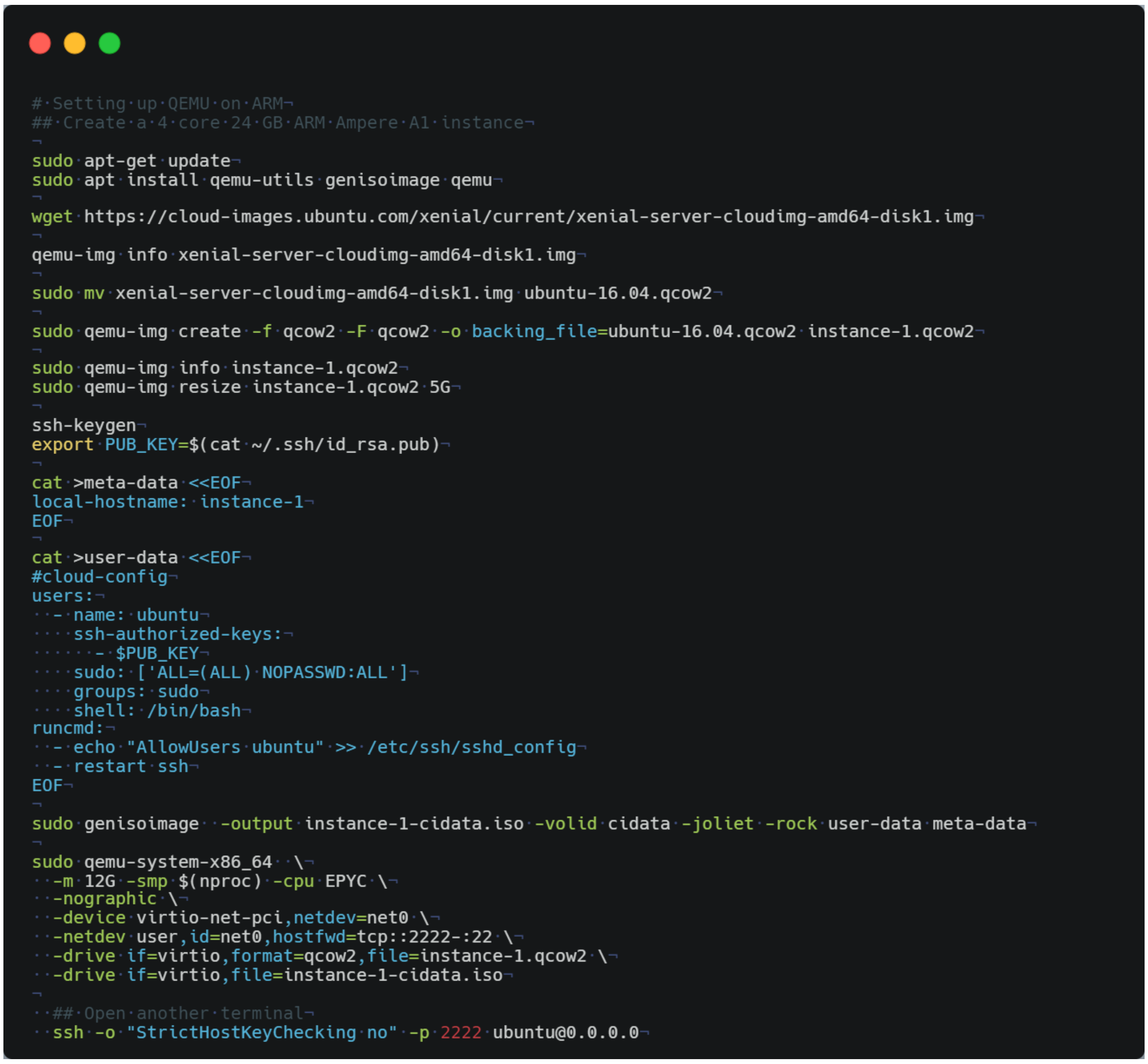
Quoting the Linux man pages for KVM:

"KVM is a virtualisation solution that is built-in to the Linux kernel. It consists of a loadable kernel module, kvm.ko, that provides the core virtualisation infrastructure and a processor-specific module, kvm-intel.ko or kvm-amd.ko. Using KVM, one can run multiple virtual machines running unmodified Linux or Windows images. Each virtual machine has private virtualised hardware: a network card, disk, graphics adapter, etc. KVM is open-source software. The kernel component of KVM is included in mainline Linux as of 2.6.20. The user-space component of KVM is included in mainline QEMU, as of 1.3 [26]."

User-level processes can also be emulated by QEMU, enabling the use of programs built for one architecture on another [27]. QEMU has previously been used to emulate x86, although at the cost of performance [28].

KVM vs QEMU

QEMU is a type 2 hypervisor that runs within user space and performs virtual hardware emulation. In contrast, KVM is a type 1 hypervisor that runs in kernel space, which allows a user space program access to the hardware virtualisation features of various processors[29].



```
# Setting up QEMU on ARM
## Create a 4 core 24 GB ARM Ampere A1 instance

sudo apt-get update
sudo apt install qemu-utils genisoimage qemu

wget https://cloud-images.ubuntu.com/xenial/current/xenial-server-cloudimg-amd64-disk1.img

qemu-img info xenial-server-cloudimg-amd64-disk1.img

sudo mv xenial-server-cloudimg-amd64-disk1.img ubuntu-16.04.qcow2

sudo qemu-img create -f qcow2 -F qcow2 -o backing_file=ubuntu-16.04.qcow2 instance-1.qcow2

sudo qemu-img info instance-1.qcow2
sudo qemu-img resize instance-1.qcow2 5G

ssh-keygen
export PUB_KEY=$(cat ~/.ssh/id_rsa.pub)

cat >meta-data <<EOF
local-hostname: instance-1
EOF

cat >user-data <<EOF
#cloud-config
users:
  - name: ubuntu
    ssh-authorized-keys:
      - $PUB_KEY
    sudo: ['ALL=(ALL) NOPASSWD:ALL']
    groups: sudo
    shell: /bin/bash
runcmd:
  - echo "AllowUsers ubuntu" >> /etc/ssh/sshd_config
  - restart ssh
EOF

sudo genisoimage  -output instance-1-cidata.iso -volid cidata -joliet -rock user-data meta-data

sudo qemu-system-x86_64  \
  -m 12G -smp $(nproc) -cpu EPYC \
  -nographic \
  -device virtio-net-pci,netdev=net0 \
  -netdev user,id=net0,hostfwd=tcp::2222-:22 \
  -drive if=virtio,format=qcow2,file=instance-1.qcow2 \
  -drive if=virtio,file=instance-1-cidata.iso

  ## Open another terminal
  ssh -o "StrictHostKeyChecking no" -p 2222 ubuntu@0.0.0.0
```

Figure 13 Setting up QEMU on ARM to emulate x86

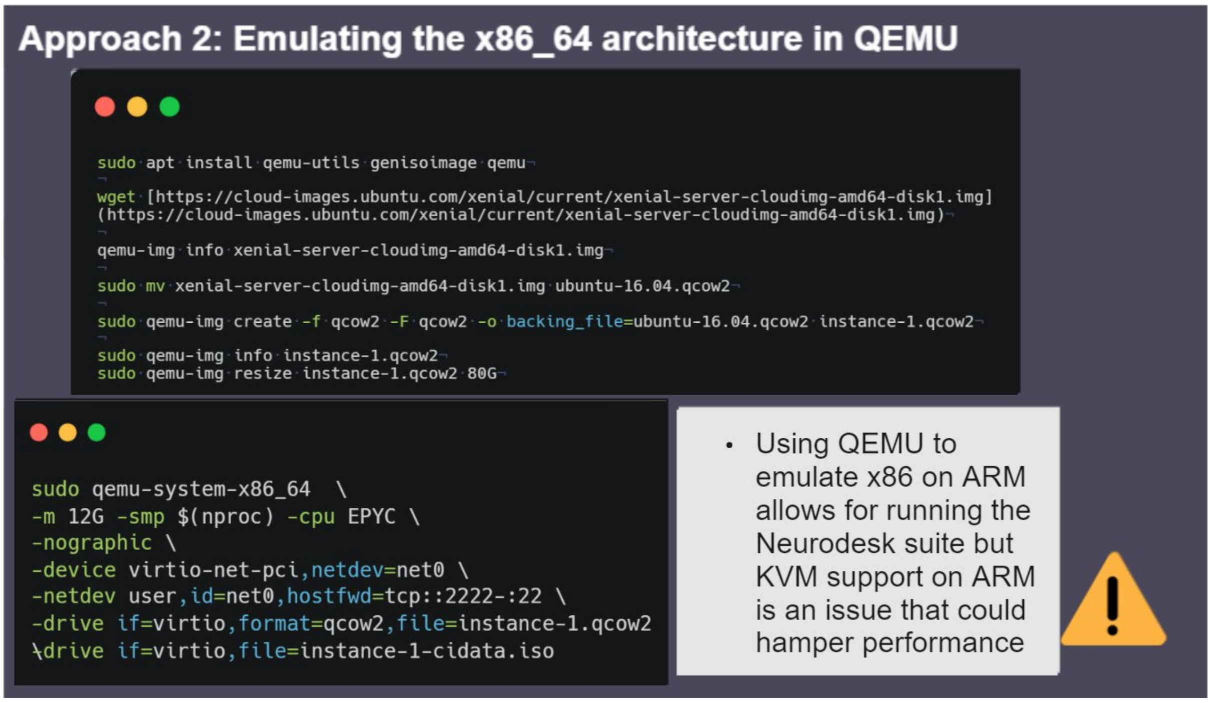
Performance while using QEMU emulation was extremely slow. The reason for this performance drop is that in order to emulate an x86_64 Linux system, QEMU must also emulate storage devices, console output devices, ethernet devices, keyboards, and the entire

CPU. This framework emulates every instruction doing everything with Just in Time translation from the Linux kernel down to your /bin/ls command [42].

ARM on Oracle Cloud has no support for KVM, which would significantly improve performance. For reference, a benchmark test of emulating ARM_64 on x86 While having enabled KVM (also x86 emulation on x86) and the performance is close to native.

Apple's Rosetta 2 tackles the problem differently - the emulation happens before the application launches. The entire binary is translated from x86 to Apple Silicon and launched. Once translated, the application is, in effect, a native arm64 binary making native macOS system calls [42]–[44].

Summary for Approach 2: Using QEMU to emulate x86 on ARM allows for running the Neurodesk suite, but KVM support on ARM is an issue that could hamper performance. QEMU emulation works for the whole NeuroDesk suite but is extremely slow.



Approach 2: Emulating the x86_64 architecture in QEMU

```
sudo apt install qemu-utils genisoimage qemu

wget [https://cloud-images.ubuntu.com/xenial/current/xenial-server-cloudimg-amd64-disk1.img]
(https://cloud-images.ubuntu.com/xenial/current/xenial-server-cloudimg-amd64-disk1.img)

qemu-img info xenial-server-cloudimg-amd64-disk1.img

sudo mv xenial-server-cloudimg-amd64-disk1.img ubuntu-16.04.qcow2

sudo qemu-img create -f qcow2 -F qcow2 -o backing_file=ubuntu-16.04.qcow2 instance-1.qcow2

sudo qemu-img info instance-1.qcow2
sudo qemu-img resize instance-1.qcow2 80G
```

```
sudo qemu-system-x86_64  \
-m 12G -smp $(nproc) -cpu EPYC \
-nographic \
-device virtio-net-pci,netdev=net0 \
-netdev user,id=net0,hostfwd=tcp::2222-:22 \
-drive if=virtio,format=qcow2,file=instance-1.qcow2
\drive if=virtio,file=instance-1-cidata.iso
```

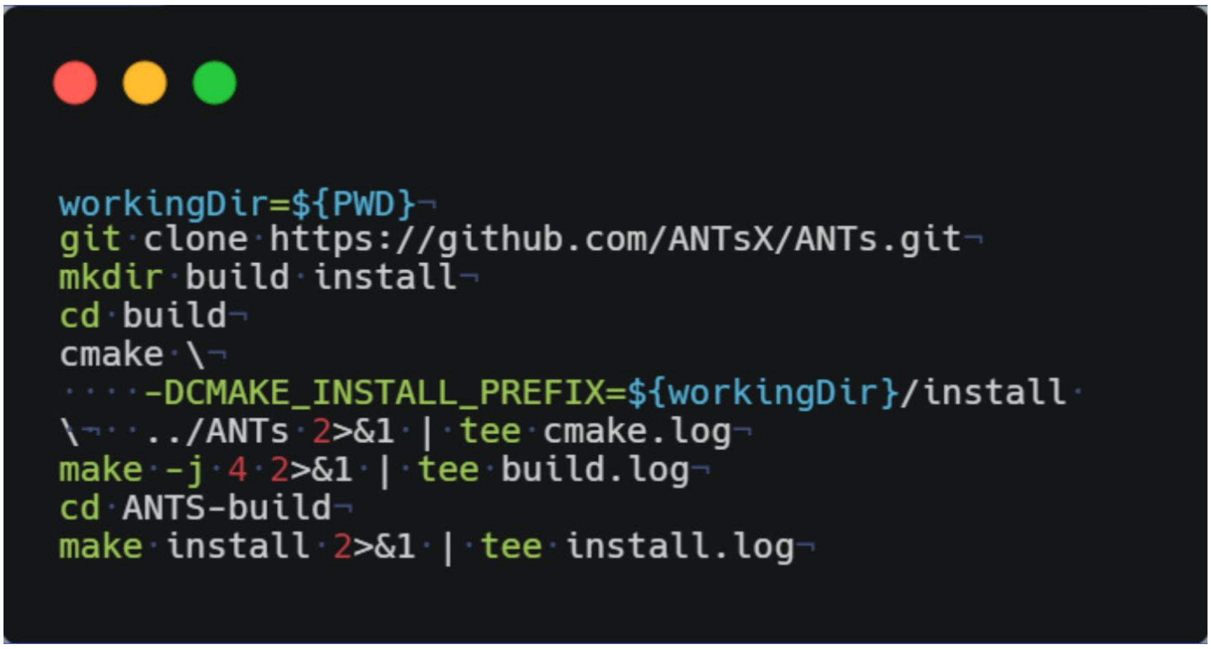
- Using QEMU to emulate x86 on ARM allows for running the Neurodesk suite but KVM support on ARM is an issue that could hamper performance

Figure 14 Approach 2: Emulating the x86_64 architecture in QEMU

Approach 3: Compiling ANTs from source on ARM64

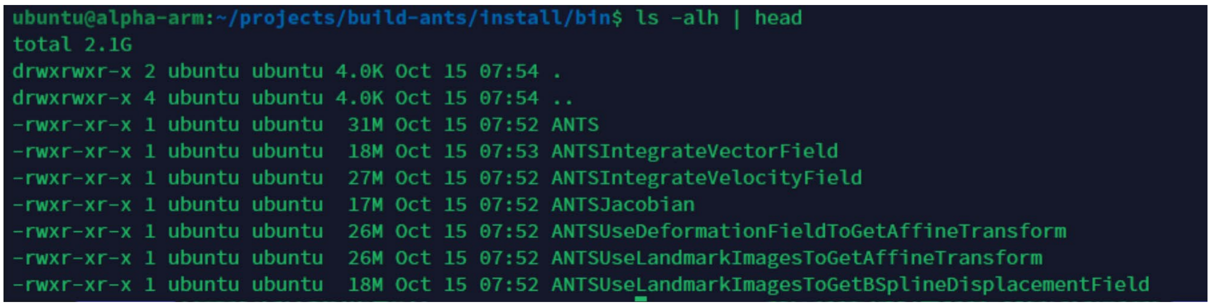
In the final approach, the idea was to get ANTs running on ARM by compiling it natively.

1. Cloning the source of ANTs and compiling natively on ARM



```
workingDir=${PWD}
git clone https://github.com/ANTsX/ANTs.git
mkdir build install
cd build
cmake \
    -DCMAKE_INSTALL_PREFIX=${workingDir}/install \
    ../ANTs 2>&1 | tee cmake.log
make -j 4 2>&1 | tee build.log
cd ANTS-build
make install 2>&1 | tee install.log
```

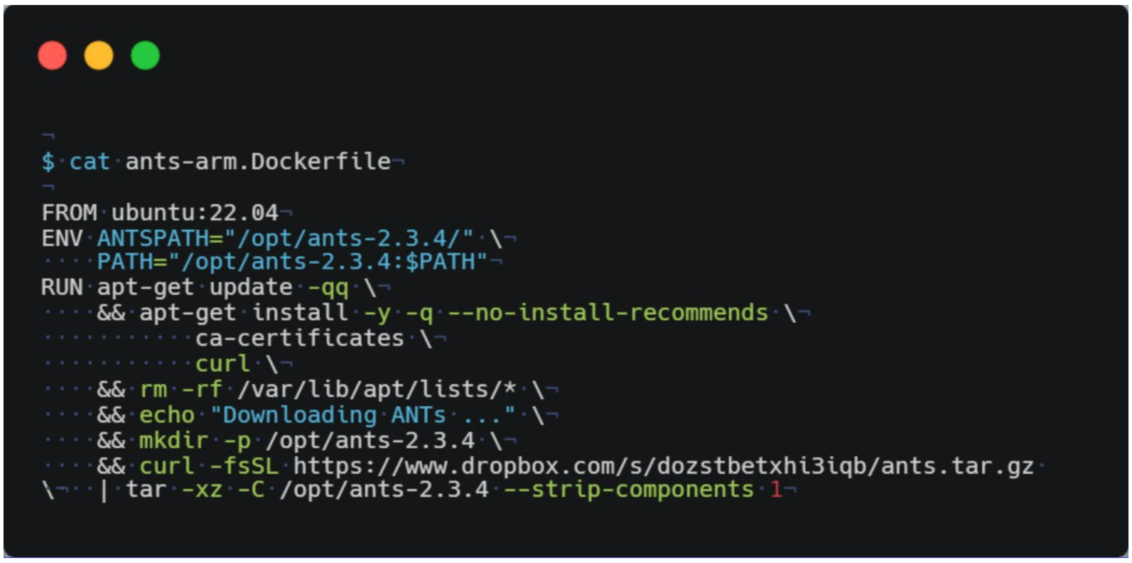
Figure 15 Compiling ANTs natively on ARM



```
ubuntu@alpha-arm:~/projects/build-ants/install/bin$ ls -alh | head
total 2.1G
drwxrwxr-x 2 ubuntu ubuntu 4.0K Oct 15 07:54 .
drwxrwxr-x 4 ubuntu ubuntu 4.0K Oct 15 07:54 ..
-rwxr-xr-x 1 ubuntu ubuntu  31M Oct 15 07:52 ANTS
-rwxr-xr-x 1 ubuntu ubuntu  18M Oct 15 07:53 ANTSIntegrateVectorField
-rwxr-xr-x 1 ubuntu ubuntu  27M Oct 15 07:52 ANTSIntegrateVelocityField
-rwxr-xr-x 1 ubuntu ubuntu  17M Oct 15 07:52 ANTSJacobian
-rwxr-xr-x 1 ubuntu ubuntu  26M Oct 15 07:52 ANTSUseDeformationFieldToGetAffineTransform
-rwxr-xr-x 1 ubuntu ubuntu  26M Oct 15 07:52 ANTSUseLandmarkImagesToGetAffineTransform
-rwxr-xr-x 1 ubuntu ubuntu  18M Oct 15 07:52 ANTSUseLandmarkImagesToGetBSplineDisplacementField
```

Figure 16 The ANTs ARM binaries after successful compilation

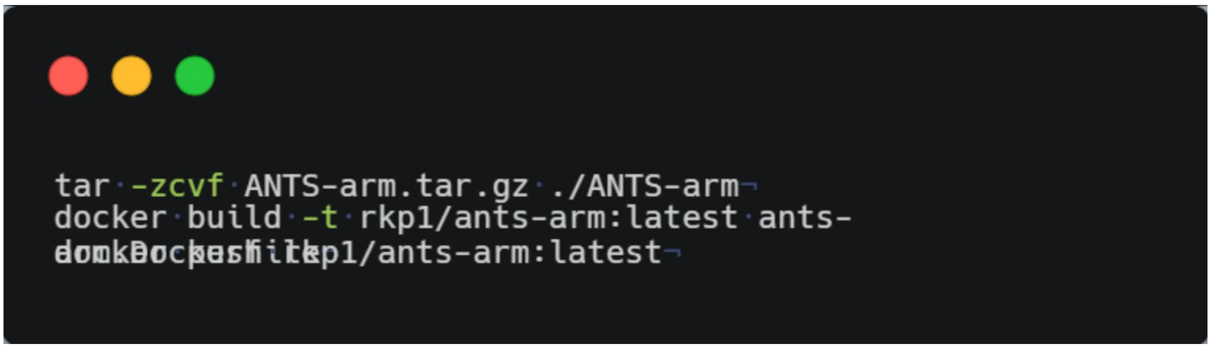
2. Zipping the compiled binaries and uploading them to Dropbox and modifying the docker file to replace the default Dropbox link with the Dropbox link with ARM binaries



```
$ cat ants-arm.Dockerfile

FROM ubuntu:22.04
ENV ANTSPATH="/opt/ants-2.3.4/" \
    PATH="/opt/ants-2.3.4:$PATH"
RUN apt-get update -qq \
    && apt-get install -y -q --no-install-recommends \
           ca-certificates \
           curl \
    && rm -rf /var/lib/apt/lists/* \
    && echo "Downloading ANTs ..." \
    && mkdir -p /opt/ants-2.3.4 \
    && curl -fsSL https://www.dropbox.com/s/dozstbetxhi3iqb/ants.tar.gz 
\  | tar -xz -C /opt/ants-2.3.4 --strip-components 1
```

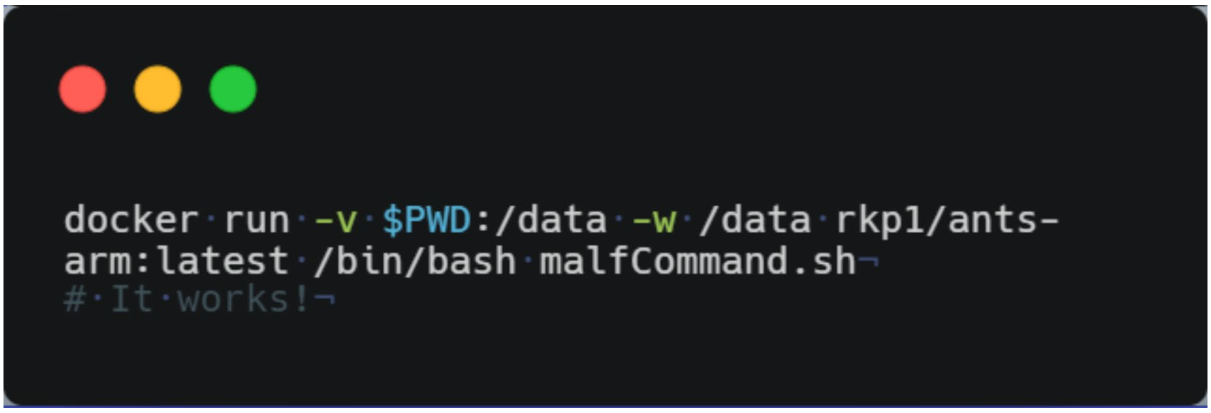
Figure 17 The modified Dockerfile for ARM



```
tar -zcvf ANTS-arm.tar.gz ./ANTS-arm
docker build -t rkp1/ants-arm:latest ants-
arm.Dockerfile
docker push rkp1/ants-arm:latest
```

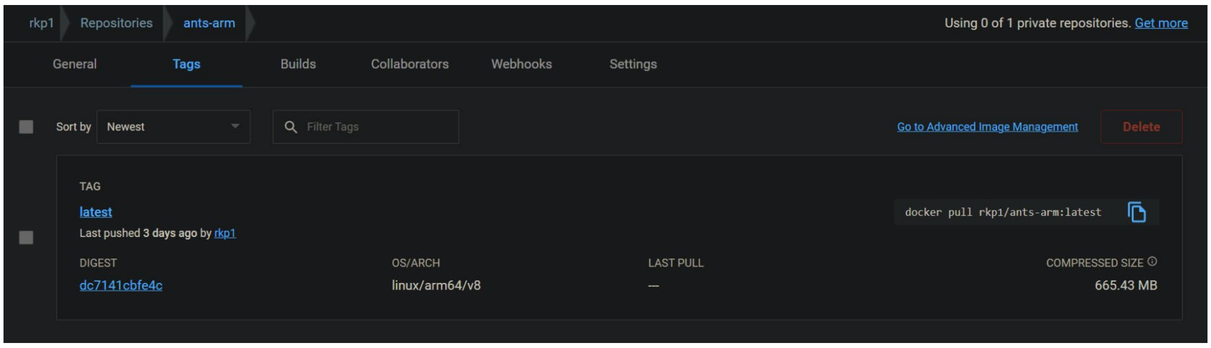
Figure 18 Zipping up the ARM binaries before uploading them to Dropbox.

3. Building the Docker Image and pushing it to DockerHub. This custom image can be accessed at https://hub.docker.com/repository/docker/rkp1/ants-arm



```
docker run -v $PWD:/data -w /data rkp1/ants-
arm:latest /bin/bash malfCommand.sh
# It works!
```

Figure 19 Building the docker image for ARM and pushing it to DockerHub.


rkp1 › Repositories › ants-arm

Using 0 of 1 private repositories. Get more

General | Tags | Builds | Collaborators | Webhooks | Settings

Sort by Newest | Filter Tags | Go to Advanced Image Management | Delete

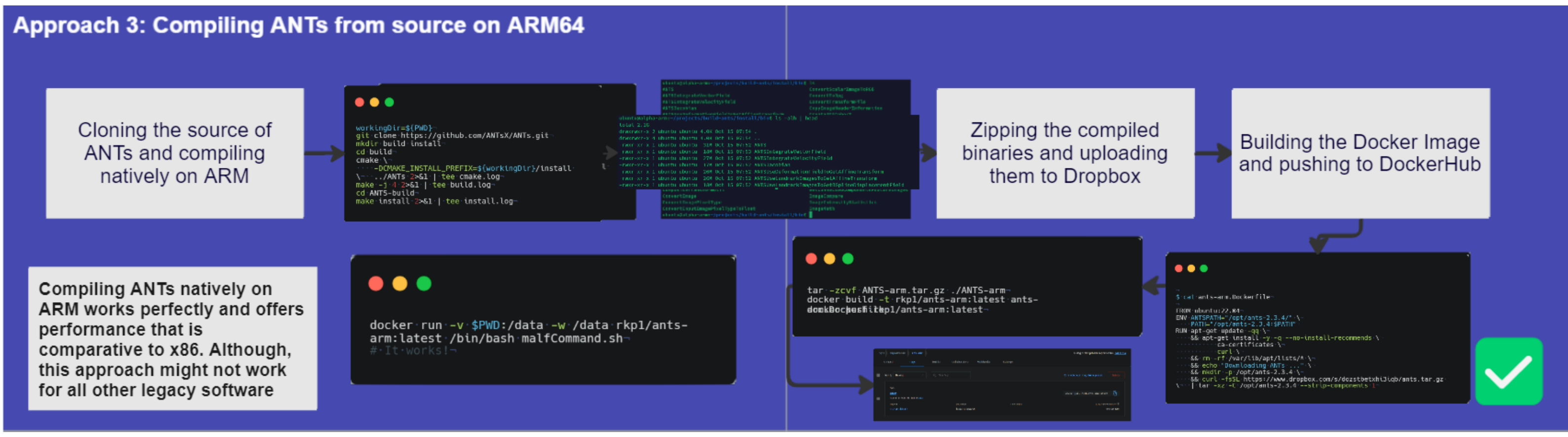
TAG
latest
Last pushed 3 days ago by rkp1

```
docker pull rkp1/ants-arm:latest
```

DIGEST	OS/ARCH	LAST PULL	COMPRESSED SIZE
dc7141cbfe4c	linux/arm64/v8	---	665.43 MB

Figure 20 The newly created Docker Image for ARM on DockerHub

Summary for Approach 3: **Compiling ANTs natively on ARM works perfectly and offers performance that is comparable to x86. Although, this approach might not work for some other legacy software.**



Approach 3: Compiling ANTs from source on ARM64
Cloning the source of ANTs and compiling natively on ARM
Zipping the compiled binaries and uploading them to Dropbox
Building the Docker Image and pushing to DockerHub

Compiling ANTs natively on ARM works perfectly and offers performance that is comparative to x86. Although, this approach might not work for all other legacy software

```
workingDir=${PWD}
git clone https://github.com/ANTsX/ANTs.git
mkdir build install
cd build
cmake \
    -DCMAKE_INSTALL_PREFIX=${workingDir}/install
\   ../ANTs 2>&1 | tee cmake.log
make -j 4 2>&1 | tee build.log
cd ANTS-build
make install 2>&1 | tee install.log
```

```
$ cat ants-arm.Dockerfile

FROM ubuntu:22.04
ENV ANTSPATH="/opt/ants-2.3.4/" \
    PATH="/opt/ants-2.3.4:$PATH"
RUN apt-get update -qq \
    && apt-get install -y -q --no-install-recommends \
           ca-certificates \
           curl \
    && rm -rf /var/lib/apt/lists/* \
    && echo "Downloading ANTs ..." \
    && mkdir -p /opt/ants-2.3.4 \
    && curl -fsSL https://www.dropbox.com/s/dozstbetxhi3iqb/ants.tar.gz
\   | tar -xz -C /opt/ants-2.3.4 --strip-components 1
```

```
tar -zcvf ANTS-arm.tar.gz ./ANTS-arm
docker build -t rkp1/ants-arm:latest ants-
arm.Dockerfile
docker push rkp1/ants-arm:latest
```

```
docker run -v $PWD:/data -w /data rkp1/ants-
arm:latest /bin/bash malfCommand.sh
# It works!
```

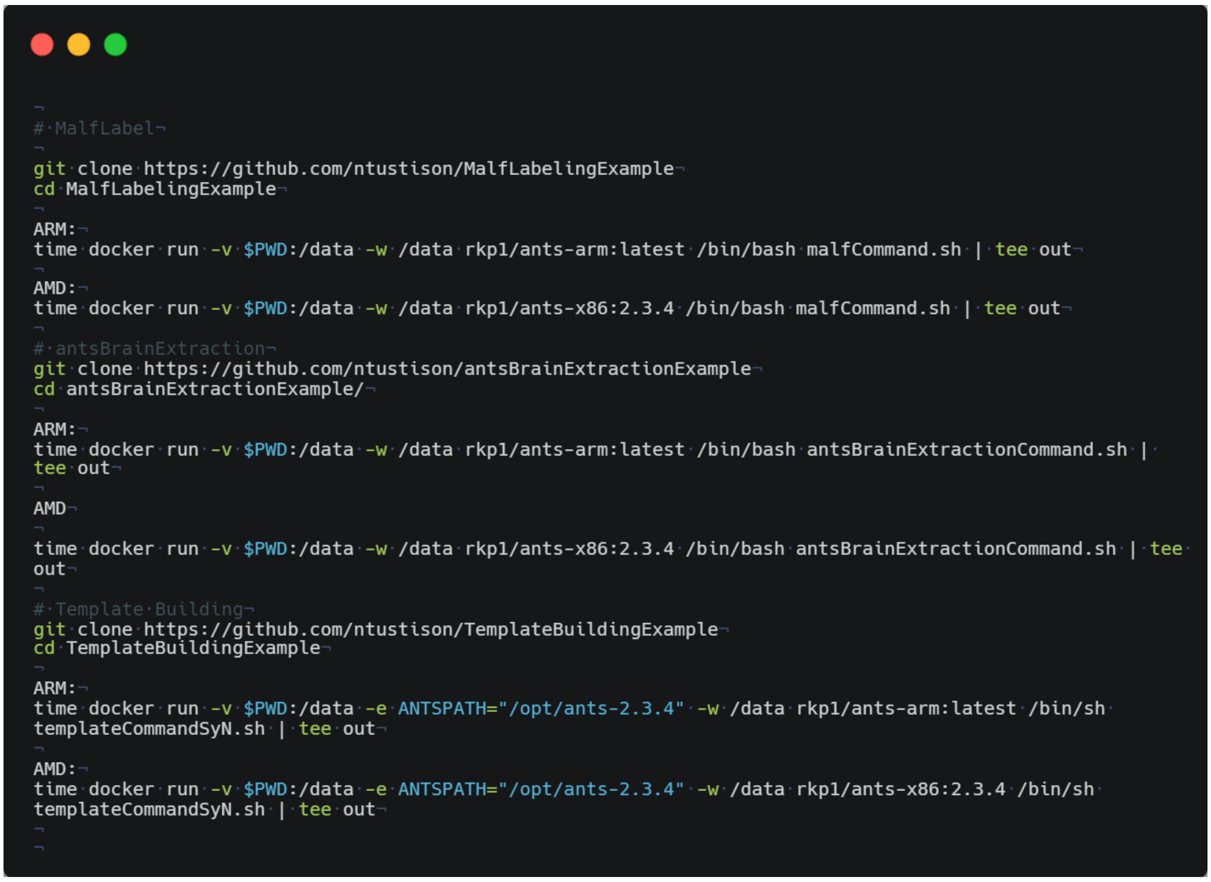
Figure 21 Approach 3: Compiling ANTs from source on ARM64

Running the benchmarks

Four example workloads from the stnava/ANTsTutorial repository were chosen. These were:

1. **ntustison/antsBrainExtractionExample** [45]
2. ntustison/**MalfLabelingExample** [46]
3. ntustison/**TemplateBuildingExample** [47]
4. ntustison/**ProtonCtLungMaskRegistration** [48]

Each benchmark was run five times on each platform, and the time taken to run each workload was measured and averaged across the five runs.



```
# MalfLabel

git clone https://github.com/ntustison/MalfLabelingExample
cd MalfLabelingExample

ARM:
time docker run -v $PWD:/data -w /data rkp1/ants-arm:latest /bin/bash malfCommand.sh | tee out

AMD:
time docker run -v $PWD:/data -w /data rkp1/ants-x86:2.3.4 /bin/bash malfCommand.sh | tee out

# antsBrainExtraction
git clone https://github.com/ntustison/antsBrainExtractionExample
cd antsBrainExtractionExample/

ARM:
time docker run -v $PWD:/data -w /data rkp1/ants-arm:latest /bin/bash antsBrainExtractionCommand.sh |
tee out

AMD

time docker run -v $PWD:/data -w /data rkp1/ants-x86:2.3.4 /bin/bash antsBrainExtractionCommand.sh | tee
out

# Template Building
git clone https://github.com/ntustison/TemplateBuildingExample
cd TemplateBuildingExample

ARM:
time docker run -v $PWD:/data -e ANTSPATH="/opt/ants-2.3.4" -w /data rkp1/ants-arm:latest /bin/sh
templateCommandSyN.sh | tee out

AMD:
time docker run -v $PWD:/data -e ANTSPATH="/opt/ants-2.3.4" -w /data rkp1/ants-x86:2.3.4 /bin/sh
templateCommandSyN.sh | tee out
```

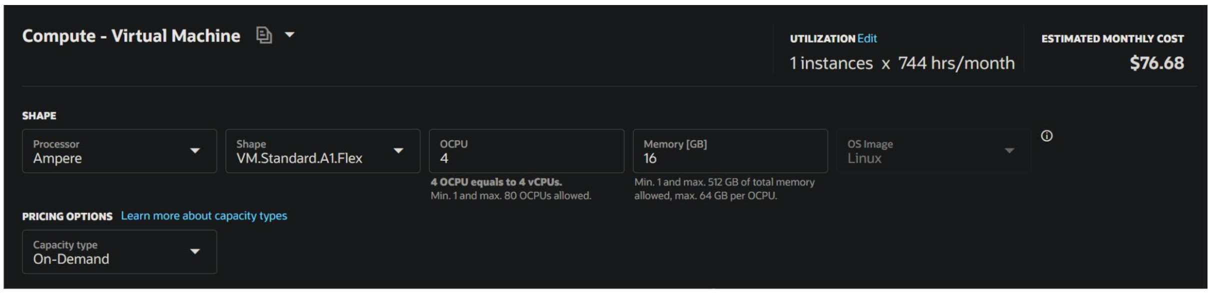
Figure 22 Running the benchmarks on ARM and x86

The ARM and x86 instances were configured to cost the same. These were deployed in the Australia/Sydney region on Oracle Cloud.

The ARM instance was configured with 4 OCPUs and 16 GB of memory, costing $76.88 per month. The x86 configuration had 2 OCPUs and 16 GB of memory for a similar cost.

For the benchmarks, the QEMU instance running x86 on ARM was configured with the same specs - 4 OCPUs and 16 GB of memory, costing the same $76.88 per month.

Another ARM instance with 2 OCPUS and 8 GB of memory was also deployed, costing $35.56 per month.



Compute - Virtual Machine

UTILIZATION Edit
1 instances x 744 hrs/month

ESTIMATED MONTHLY COST
$76.68

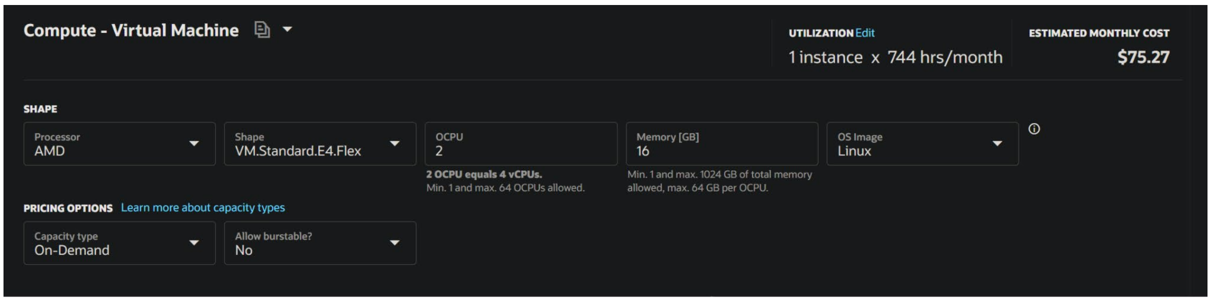
SHAPE

Processor: Ampere
Shape: VM.Standard.A1.Flex
OCPU: 4
4 OCPU equals to 4 vCPUs.
Min. 1 and max. 80 OCPUs allowed.
Memory [GB]: 16
Min. 1 and max. 512 GB of total memory allowed, max. 64 GB per OCPU.
OS Image: Linux

PRICING OPTIONS Learn more about capacity types

Capacity type: On-Demand

Figure 23 ARM instance configuration on Oracle Cloud



Compute - Virtual Machine

UTILIZATION Edit
1 instance x 744 hrs/month

ESTIMATED MONTHLY COST
$75.27

SHAPE

Processor: AMD
Shape: VM.Standard.E4.Flex
OCPU: 2
2 OCPU equals 4 vCPUs.
Min. 1 and max. 64 OCPUs allowed.
Memory [GB]: 16
Min. 1 and max. 1024 GB of total memory allowed, max. 64 GB per OCPU.
OS Image: Linux

PRICING OPTIONS Learn more about capacity types

Capacity type: On-Demand
Allow burstable?: No

Figure 24 x86 instance configuration on Oracle Cloud