Get ready for Arm64
.. and an ARMful of container build cases

Syl Taylor

Specialist Solutions Architect

Compute

Ovidiu Valeanu

Specialist Solutions Architect

Containers

Arm-based compute options

On-premises and Edge

- Vendor servers (e.g. GIGABYTE)
- Apple MacBook with M1 chips
- Ampere Workstations
- Raspberry Pi >= 3
- Mobile phones & IoT devices
- Supercomputers (e.g. Fujitsu A64FX)

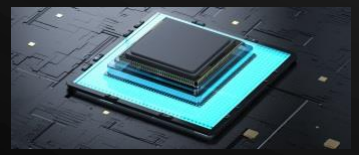
Cloud

- AWS Graviton, Ampere Altra, Alibaba Yitian, etc.
- Bare-metal, VMs, and managed services

Over 230 billion Arm chips produced

Running software on arm64

- Applications
- Libraries & Frameworks
- Container Images
- Container Runtime
- Operating System
- Hypervisor



Entire software stack

must be compatible

with the CPU architecture (arm64)

arm64 is also known as aarch64

Different CPU architectures

x86_64

```
mov     DWORD PTR [rbp-4], 10
mov     DWORD PTR [rbp-8], 20
mov     edx, DWORD PTR [rbp-4]
mov     eax, DWORD PTR [rbp-8]
add     eax, edx
```

arm64

```
mov     w0, 10
str     w0, [sp, 12]
mov     w0, 20
str     w0, [sp, 8]
ldr     w1, [sp, 12]
ldr     w0, [sp, 8]
add     w0, w1, w0
```

different ISA, different instructions

```
int sum() {
    int a = 10, b = 20;
    return a + b;
}
```

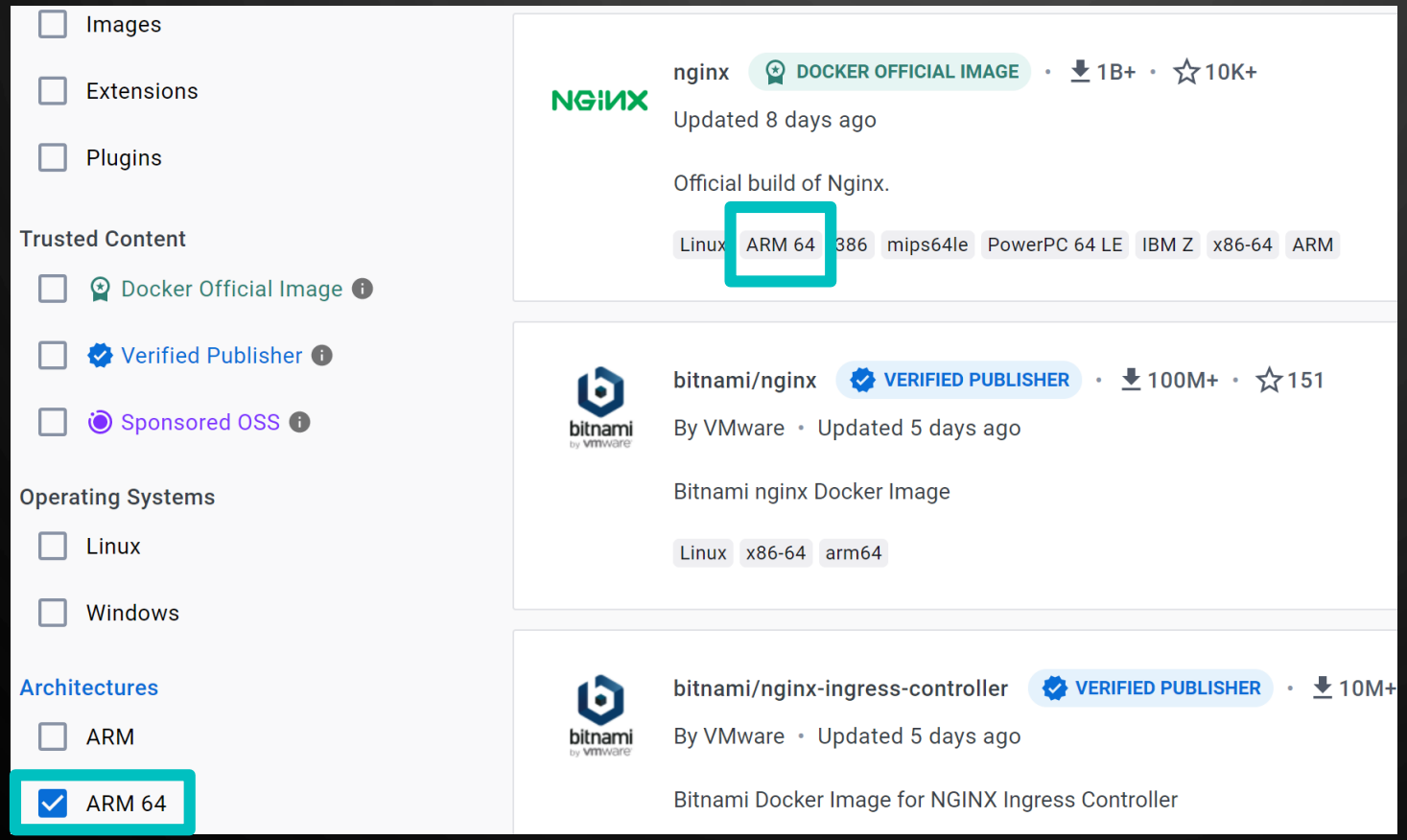
same code

-O0 (no optimizations)

Select a compatible base image

Use empty images: “FROM scratch”

OR



Registries will use a tag like “ARM 64” with e.g. OS/ARCH: “linux/arm64/v8”

Multi-arch container images

```
$ docker manifest inspect nginx:1.23.3
{
  "schemaVersion": 2,
  "mediaType": "application/vnd.docker.distribution.manifest.list.v2+json",
  "manifests": [
    {
      "mediaType": "application/vnd.docker.distribution.manifest.v2+json",
      "size": 1570,
      "digest": "sha256:942ae2dfd73088b54d7151a3c3fd5af038a51c50029bfcfd21f1e650d9579967",
      "platform": {
        "architecture": "amd64",
        "os": "linux"
      }
    },
    {
      "mediaType": "application/vnd.docker.distribution.manifest.v2+json",
      "size": 1570,
      "digest": "sha256:d415dd4e87d75f4d6607340d6d6fad78b0ed66b9c809eedb79ff08f3e58d008f",
      "platform": {
        "architecture": "arm64",
        "os": "linux",
        "variant": "v8"
      } ...
```

Running an arm64 container image

On an arm64 machine:

```
$ docker pull nginx:1.23.3

$ docker image inspect nginx:1.23.3 | grep 'Architecture'
 "Architecture": "arm64",

$ docker run --name docker-nginx -p 127.0.0.1:1080:80 nginx:1.23.3
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
...
/docker-entrypoint.sh: Configuration complete; ready for start up
...
2023/03/10 13:27:35 [notice] 1#1: nginx/1.23.3
2023/03/10 13:27:35 [notice] 1#1: built by gcc 10.2.1 20210110 (Debian 10.2.1-6)
2023/03/10 13:27:35 [notice] 1#1: OS: Linux 5.15.0-1031-aws
2023/03/10 13:27:35 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2023/03/10 13:27:35 [notice] 1#1: start worker processes
...
```

Watch out for: exec format error

On an arm64 machine:

```
$ docker run --platform linux/amd64 nginx:1.23.3

Unable to find image 'nginx:1.23.3' locally
1.23.3: Pulling from library/nginx
3f9582a2cbe7: Already exists
9a8c6f286718: Pull complete
e81b85700bc2: Pull complete
73ae4d451120: Pull complete
6058e3569a68: Pull complete
3a1b8f201356: Pull complete

Digest: sha256:aa0afebbb3cfa473099a62c4b32e9b3fb73ed23f2a75a65ce1d4b4f55a5c2ef2

Status: Downloaded newer image for nginx:1.23.3

exec /docker-entrypoint.sh: exec format error
```

Building arm64 container images

Method #1: Natively (on an arm64 machine, *locally or remote builds*)

```
$ docker build -t example:0.1 .
```

Method #2: Emulation (on an x86_64 machine)

```
$ docker buildx build --platform linux/arm64 .
```

Method #3: Cross-compilation (on an x86_64 machine)

```
FROM --platform=linux/amd64 ubuntu as build
# RUN <install build dependencies>
# COPY <source> .
# RUN <cross-compiler for target: linux/arm64, -o /out/arm64_binary>

FROM --platform=linux/arm64 ubuntu as runtime
# RUN <install runtime dependencies via emulation>
COPY --from=build /out/arm64_binary /bin
```

Buildah and multi-arch images

- **Daemon-less, rootless, OCI multi-arch compatible image builder**
- **Build container from scratch or pre-existing Dockerfile**

```
# Create a multi-architecture manifest
buildah manifest create ${MANIFEST_NAME}

# Build your amd64 architecture container
buildah bud \ --tag "${REGISTRY}/${USER}/${IMAGE_NAME}:${IMAGE_TAG}" \ --
manifest ${MANIFEST_NAME} \ --arch amd64 \ ${BUILD_PATH}

# Build your arm64 architecture container
buildah bud \ --tag "${REGISTRY}/${USER}/${IMAGE_NAME}:${IMAGE_TAG}" \ --
manifest ${MANIFEST_NAME} \ --arch arm64 \ ${BUILD_PATH}

# Push the full manifest, with both CPU architectures
buildah manifest push --all \ ${MANIFEST_NAME} \
"docker://${REGISTRY}/${USER}/${IMAGE_NAME}:${IMAGE_TAG}"
```

Emulation sounds great! But is it?

Let's build an arm64 container image from machines with 16 vCPUs & 32 GiB

```
$ git clone https://github.com/IntelRealSense/librealsense.git
$ cd librealsense/scripts/Docker
$ LIBRS_GIT_TAG=`git describe --abbrev=0 --tags`
$ LIBRS_VERSION=${LIBRS_GIT_TAG#"v"}
```

Emulated build – from x86_64

```
$ time docker buildx build --platform linux/arm64 -t librealsense-arm64:0.1 . --build-arg LIBRS_VERSION=$LIBRS_VERSION
...    22m27.874s...
```

Native build – on arm64

```
$ time docker build -t librealsense-arm64:0.1 . --build-arg LIBRS_VERSION=$LIBRS_VERSION
...    3m1.891s
```

Emulation is slow

Let's look at Dockerfile

```
....
RUN apt-get update && \
      apt-get install -qq -y --no-install-recommends build-essential cmake git libssl-dev libusb-  1.0-0-dev pkg-config libgtk-3-dev
libglfw3-dev libgl1-mesa-dev libglu1-mesa-dev curl python3 python3-dev ca-certificates && \
....
RUN cd /usr/src/librealsense &&\
      mkdir build && cd build &&\
      cmake \
      -DCMAKE_C_FLAGS_RELEASE="${CMAKE_C_FLAGS_RELEASE} -s" \
      -DCMAKE_CXX_FLAGS_RELEASE="${CMAKE_CXX_FLAGS_RELEASE} -s" \
      -DCMAKE_INSTALL_PREFIX=/opt/librealsense \
      -DBUILD_GRAPHICAL_EXAMPLES=OFF \
      -DBUILD_PYTHON_BINDINGS:bool=true \
      -DCMAKE_BUILD_TYPE=Release ../ &&\
      make -j$(($(nproc)-1)) all && \
      make install
...
```

Note the RUN commands which during emulation will require instruction translation

But if we stick to native builds, then..

Case 1: No changes needed (build)

Dockerfile

```
FROM public.ecr.aws/lts/ubuntu:20.04_stable

WORKDIR /home/app
COPY requirements.txt .

RUN apt update && \
      apt install -y --no-install-recommends python3 python3-pip && \
      pip install -r requirements.txt
```

requirements.txt

```
numpy==1.21.6
scikit-learn==1.2.0
matplotlib==3.6.1
```

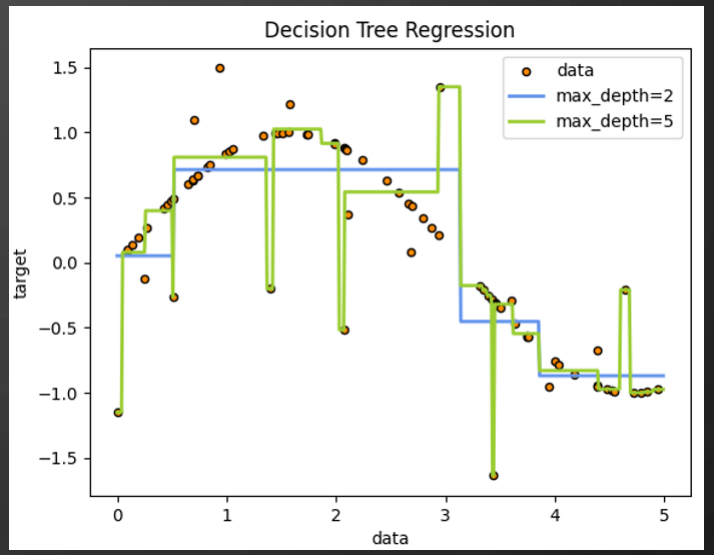
```
$ docker build -t sklearn-example:0.1 .
```

Case 1: No changes needed (runtime)

app.py

```
import numpy as np
from sklearn.tree import DecisionTreeRegressor
import matplotlib.pyplot as plt

# Create a random dataset
rng = np.random.RandomState(1)
...

# Fit regression model
...
regr_1.fit(X, y)
regr_2.fit(X, y)

# Predict
...
y_1 = regr_1.predict(X_test)
y_2 = regr_2.predict(X_test)

# Plot the results
...
plt.savefig("plot.png")
```

plot.png




```
$ docker run -v ./:/home/app -it sklearn-example:0.1 /usr/bin/python3 app.py
```

Case 1: No changes needed(demo)

View file: demo-case-1.mp4

Case 1: Why are no changes needed?

Move from x86_64/amd64 -> arm64 with no effort:

1. *public.ecr.aws/lts/ubuntu:20.04_stable* is a **multi-arch base image**
2. Python *pip wheels & versions* are available for both **x86_64** & **arm64**
3. Packages installed with *apt* are available for both **x86_64** & **arm64**
4. app.py is written in **pure Python** (no hardware-specific low-level code)

How about the other cases?

Watch out for:

- **Successful builds are great, but also check for runtime errors**

Build success. Runtime error.

```
 > [3/3] RUN python3 -c 'import tvm; print(tvm.__version__)':
...
#6 0.587 ModuleNotFoundError: No module named 'typing_extensions'
------
Dockerfile:7
--------------------
   5 |         pip install apache-tvm
   6 |
   7 | >>> RUN python3 -c 'import tvm; print(tvm.__version__)'
   8 |
--------------------
```

- **Validate your applications through software testing**

  (for example, CPU architecture changes can affect floating number comparisons)

Case 2: Old version issue

Dockerfile

```
FROM public.ecr.aws/lts/ubuntu:20.04_stable

WORKDIR /home/app

RUN apt update && \
      apt install -y --no-install-recommends python3-pip

RUN pip install numpy==1.18.5
```

app.py

```
import numpy as np
print(np.__version__)
print(np.arange(15).reshape(3, 5))
```

Case 2: Old version issue

Error – arm64

```
#0 11.84         raise RuntimeError("Broken toolchain: cannot link a simple C program")
#0 11.84     RuntimeError: Broken toolchain: cannot link a simple C program
```

x86_64

```
#7 [4/4] RUN pip install numpy==1.18.5
#7 1.548 Collecting numpy==1.18.5
#7 1.563   Downloading numpy-1.18.5-cp38-cp38-
manylinux1_x86_64.whl (20.6 MB)
#7 2.141 Installing collected packages: numpy
#7 3.538 Successfully installed numpy-1.18.5
#7 DONE 3.8s
```

arm64

```
#7 [4/4] RUN pip install numpy==1.18.5
#7 1.278 Collecting numpy==1.18.5
#7 1.295   Downloading numpy-1.18.5.zip (5.4 MB)
#7 1.587   Installing build dependencies: started
#7 4.239   Installing build dependencies: finished with
status 'done'
#7 4.242   Getting requirements to build wheel: started
#7 4.448   Getting requirements to build wheel: finished
with status 'done'
#7 4.450     Preparing wheel metadata: started
#7 11.54     Preparing wheel metadata: finished with
status 'error'
```

no pre-built pip wheel available

Case 2: Old version issue (ideas)

Potential solutions to fix the pip wheel for numpy on arm64:

1. Install *build dependencies* and version 1.18.5 will work

```
RUN apt update && \
    apt install -y --no-install-recommends gpg-agent software-properties-common python3-pip build-essential && \
    add-apt-repository -y ppa:deadsnakes/ppa && \
    apt install -y python3.7 python3.7-dev python3.7-distutils && \
    update-alternatives --install /usr/bin/python3 python3 /usr/bin/python3.7 1
```

2. Keep the current Dockerfile, but install version >= 1.19.0

3. Use different package repo like *apt install python3-numpy* (1.17.4)

Case 2: Old version issue(demo)

View file: demo-case-2.mp4

Case 3: No binary available

Dockerfile

```
FROM public.ecr.aws/lts/ubuntu:20.04_stable

RUN apt update && \
      apt install -y --no-install-recommends python3 python3-pip && \
      pip install apache-tvm==0.9.0 typing-extensions

RUN python3 -c 'import tvm; print(tvm.__version__)'
```

Error – arm64

```
#5 9.707 ERROR: Could not find a version that satisfies the requirement apache-tvm==0.9.0 (from versions: none)
#5 9.707 ERROR: No matching distribution found for apache-tvm==0.9.0
```

Case 3: No binary available (build)

Change Dockerfile. Build from source. An example below:

```
FROM public.ecr.aws/lts/ubuntu:20.04_stable

ARG LLVM_VERSION="llvmorg-14.0.6"
ARG LLVM_BIN="clang+llvm-14.0.6-aarch64-linux-gnu"
ARG TVM_VERSION="v0.9.0"
ARG DLPACK_VERSION="v0.5"

RUN apt update && \
      apt install -y --no-install-recommends git wget python3 python3-pip python3-dev python3-setuptools && \
      apt install -y --no-install-recommends libtinfo-dev zlib1g-dev build-essential cmake libedit-dev libxml2-dev libncurses5 && \
      cd /tmp; wget https://github.com/llvm/llvm-project/releases/download/${LLVM_VERSION}/${LLVM_BIN}.tar.xz && \
      mkdir llvm; tar -xf ${LLVM_BIN}.tar.xz -C /tmp/llvm && \
      git clone --recursive https://github.com/apache/tvm tvm && \
      cd tvm; git checkout tags/${TVM_VERSION} && \
      cd 3rdparty/dlpack; git checkout tags/${DLPACK_VERSION}; cd ../.. && \
      mkdir build && \
      sed -i "/set(USE_LLVM OFF)/c\set(USE_LLVM /tmp/llvm/${LLVM_BIN}/bin/llvm-config)" cmake/config.cmake && \
      cp cmake/config.cmake build && \
      cd build; cmake ..; make -j$(nproc); cd .. && \
      cd python; python3 setup.py install --user; cd ..

RUN python3 -c 'import tvm; print(tvm.__version__)'
```

Case 3: No binary available(demo)

View file: demo-case-3.mp4

Case 4: Dependency issue

Dockerfile

```
FROM public.ecr.aws/lts/ubuntu:20.04_stable

RUN apt update && \
      apt install -y --no-install-recommends python3 python3-pip && \
      pip install confluent_kafka

RUN python3 -c 'import confluent_kafka; print(confluent_kafka.version())'
```

Error 1 – arm64 (trying to build from source)

```
#5 9.868   Downloading confluent-kafka-2.0.2.tar.gz (119 kB)
...
unable to execute 'aarch64-linux-gnu-gcc': No such file or directory
```

Error 2 – arm64 (dependency with specific version not found)

```
#5 21.86       66 | #error "confluent-kafka-python requires librdkafka v2.0.2 or later.
```

Case 4: Dependency issue (build)

Change Dockerfile. Build both the pip wheel and librdkafka from source. An example:

```
FROM public.ecr.aws/lts/ubuntu:20.04_stable

ARG LIBRDKAFKA_VERSION="2.0.2"

RUN apt update && \
      apt install -y --no-install-recommends git build-essential python3 python3-pip python3-dev && \
      git clone -b v${LIBRDKAFKA_VERSION} https://github.com/confluentinc/librdkafka && \
      cd librdkafka && ./configure --install-deps && \
      make && \
      make install && \
      ldconfig && \
      pip install confluent-kafka==2.0.2 --no-binary :all:

RUN python3 -c 'import confluent_kafka; print(confluent_kafka.version())'
```

Case 4: Dependency issue(demo)

View file: demo-case-4.mp4

Case 5: Application issue

Dockerfile

```
FROM public.ecr.aws/lts/ubuntu:20.04_stable

WORKDIR /home/app
# Install certificates for github and proxy.golang

RUN apt update && \
      apt install -y --no-install-recommends wget git && \
      wget --no-check-certificate https://go.dev/dl/go1.20.2.linux-amd64.tar.gz && \
      rm -rf /usr/local/go && tar -C /usr/local -xzf go1.20.2.linux-amd64.tar.gz && \
      export PATH=$PATH:/usr/local/go/bin && \
      git clone https://github.com/segmentio/parquet-go.git && \
      cd parquet-go && \
      go test -tags amd64 -v ./
```

Error – arm64

```
...go/src/crypto/internal/boring/sig/sig_amd64.s:38: unrecognized instruction "BYTE"
...go/.../klauspost/compress@v1.15.9/internal/cpuinfo/cpuinfo_amd64.s:25: unrecognized instruction "BTQ"
...go/.../klauspost/compress@v1.15.9/internal/cpuinfo/cpuinfo_amd64.s:26: unrecognized instruction "SETCS"
#10 29.64 asm: too many errors ...
```

Case 5: Application issue

Thanks to tags (e.g. //go:build purego || !amd64), we can still use package without the asm.

Change Dockerfile. Test purego instead of amd64 (which has non-arm64 assembly .s files) .

```
FROM public.ecr.aws/lts/ubuntu:20.04_stable

WORKDIR /home/app

# Install certificates for github and proxy.golang

RUN apt update && \
      apt install -y --no-install-recommends wget git && \
      wget --no-check-certificate https://go.dev/dl/go1.20.2.linux-amd64.tar.gz && \
      rm -rf /usr/local/go && tar -C /usr/local -xzf go1.20.2.linux-amd64.tar.gz && \
      export PATH=$PATH:/usr/local/go/bin && \
      git clone https://github.com/segmentio/parquet-go.git && \
      cd parquet-go && \
      go test -tags purego -v ./
```

Case 5: Application issue(demo)

View file: demo-case-5.mp4

Other notable cases

- **TensorFlow** for arm64 is not built with CUDA support for Nvidia GPUs
- **PostgreSQL** should be compiled with Arm's LSE (atomic instructions)
- **Emscripten** build failed when a new package was added (cipd – no linux-arm64)
- **Go compiler** binary, e.g. goX.linux-amd64.tar.gz -> goX.linux-arm64.tar.gz
- Re-compiling with **Go >= 1.18** leads to up to 20% improvement on arm64

Ecosystem

The state of the ecosystem is continuously changing, meaning more arm64 binaries and optimizations are added as we speak...

As a general rule: newer software versions tend to have arm64 support

Open-source call to action

Tell software maintainers when you need arm64/aarch64 binaries or container images.

Open a new Issue or +1 to an existing one asking for support like in this example: