Arm64 Quick Guide*

Syl Taylor

Specialist Solutions Architect

Compute

* Cheat Sheet for Cloud Compute

Table of Contents

Arm64 Fundamentals

Appendix

CPU Architecture

Processors

- A processor is the most essential part of a compute device
- It processes instructions that tell parts of a computer what to do

Naming

- Arm’s 64-bit CPUs: **arm64** / **aarch64**
- Arm’s 32-bit CPUs: **armhf** / **aarch32**

Ecosystem

- The Arm CPU architecture (32-bit and 64-bit) is implemented in a large number of devices **(>230 billions**)
- Traditionally, Arm-based CPUs were found in embedded devices, such as mobile phones and IoT machines
- **Arm’s 64-bit CPUs enable all devices** (e.g. servers, laptops), providing **performant** & **power-efficient computing**

Software

- The **entire software stack** (from hypervisor to application) must be compatible with the host’s CPU arch. (e.g. **arm64**)

Cloud

- Cloud compute offerings use the **Arm64** CPU architecture. Some also support running Arm 32-bit apps

Compute Options

Cloud

- AWS Graviton (in Amazon EC2 and many other AWS Services) – **pioneered Arm64 cloud servers in 2018**
- Ampere Altra (used by most major cloud providers, e.g. Microsoft Azure, GCP, OCI)
- Alibaba Yitian, Apple M1 (such as EC2 M1 Mac instances on AWS), etc.

Servers

- Vendors (e.g. GIGABYTE) supply **Arm64** servers for on-premises based on e.g. Ampere Altra processors
- Supercomputers such as Fujitsu A64FX

examples

Laptops / Desktops

- Apple MacBooks with M1 chips (starting 2020)
- Windows on Arm laptops (multiple vendors)
- Ampere Workstations

Edge Devices

- Most mobile phones post 2010s
- Most modern IoT devices
- Raspberry Pi >= 3

Operating Systems

Cloud

- Choose a supported 64-bit OS compiled for **arm64**/**aarch64**
- On AWS, you can use Linux-based AMIs for 64-bit Arm (e.g. debian-11-**arm64**-20220503-998)
- CPUs based on Arm Neoverse N1 and V1 can't run 32-bit Arm OS or hypervisors (only 64-bit Arm)

Downloads

- Ubuntu Server for Arm: ubuntu-22.04.2-live-server-**arm64**.iso

examples

On A Linux Host

```
$ uname -a
Linux … 5.10.0-14-cloud-arm64 #1 SMP Debian 5.10.113-1 (2022-04-29) aarch64 GNU/Linux
$ uname -m
aarch64
$ sudo apt -y install linux-image-5.10.0-14-rt-arm64 (change kernel)
```

Running Software

Overview

- For applications to run on a computer, they must be compatible with the underlying CPU architecture (e.g. **arm64**)
- If you can't find **arm64** binaries to install, tell software maintainers or vendors you need them
- In general, **newer software versions** have better **arm64** support (e.g. binaries, optimizations)

Binaries

- Example of a standalone binary: go1.20.3.linux-**arm64**.tar.gz
- Example of a pip wheel: numpy-1.24.2-cp310-cp310-manylinux_2_17_aarch64.manylinux2014_**aarch64**.whl
- Example of a container image: public.ecr.aws/nginx/nginx:1.22.1-**arm64**v8

Languages

- Interpreted (e.g. Python, Ruby) or byte-code (e.g. Java) pure (non-native) code requires no changes
- Compiled (e.g. C, C++, Go, Rust) code will need to be re-compiled
- Hardware dependent code (e.g. intrinsic functions, assembly) will need to be re-written (ported)

Coding Languages

Overview

- Most cases will not require code changes to move from x86_64 to **arm64** (to run code successfully)
- Hardware-specific optimizations (e.g. assembly) in code will need to be re-written
- Some code might be inefficient on **arm64** and will need changes to optimize it

Ecosystem *examples*

- Run as normal (pure non-native code): Python, Java, Ruby, PHP, JavaScript on server side, etc.
- Re-compile or re-build (code changes might be needed): Python extensions, Java Native Interface, Go extensions, etc.
- Re-compile or re-build (code changes might be needed): C, C++, Go, Rust, etc.

Caveats

- Dependencies/modules/libraries (if not supported on **arm64**) will require additional effort
- Newer language versions (and associated runtimes, interpreters, etc.) are far more likely to perform well on **arm64**

Handling Dependencies

Overview

- The hardest parts of moving from x86_64 to **arm64** is 1) code changes (if applicable) and 2) resolving dependencies

When Issues Arise

Upgrade:

- Check if newer version has **arm64** support
- Check if package has an upgrade path
- Use tests to check for breaking changes

Replace:

- Look for alternative packages with similar features
- Use tests to check for breaking changes
- Build from source for **arm64** and link to it

Remove:

- Take out unused dependencies & reduce technical debt
- Duplicate code and fix, then maintain separate version

Support:

- Leverage open-source communities for help
- The **arm64** ecosystem is growing

Deprecate:

- Sometimes re-writing a dependency to enable it to run on **arm64** is the remaining option

Performance Testing

Setup

- Use comparable machines (processor generation, number of CPU cores, memory size, etc.)
- Each workload will need different tooling (e.g. load generators, data generators, extra code) to measure performance
- Use profiling tools and follow best practices for performance analysis (similar to other CPU architectures)

Tips

- Don’t rely on benchmarks, and instead measure specific workloads for accurate numbers
- Arm64 servers don’t use SMT. Some multi-threaded workloads will have more consistent or higher performance
- Maximize CPU usage on comparable machines, then measure a **relevant metric** (e.g. reads/sec, completion time)

Caveats

- Some workloads are not compute-bound (use minimal CPU resources)
- Older software versions might not have **arm64** performance optimizations

Optimizations

SIMD

- To improve performance for some workloads (e.g. HPC, ML), use SIMD for parallel processing
- Arm has 2 SIMD options: NEON and SVE. Check CPU spec for SIMD support

NEON

- Intrinsics
- Assembly

SVE

- Intrinsics
- Assembly

examples

LSE

- LSE atomics instructions can improve multi-threaded performance on Arm. Use e.g. -moutline-atomics (compiler option)

Compilers

- Use flags for the host architecture and experiment with the options

DevOps

Overview

- Most DevOps tooling will have support for **arm64**. Check the tool's documentation for instructions
- Recommended approach is to use (native) **arm64** runners for building and testing
- Hardest part will be ensuring e.g. software builds and tests work on **arm64** (refer to previous slides on software)

Emulation

- A convenient option is to use existing x86_64 runners to build & test software for **arm64**
- However, emulation for compute-intensive tasks is very slow and it can also introduce difficult bugs

Tools

- GitLab CI/CD
- GitHub Actions
- CircleCI
- Jenkins
- BuildKite

examples

- AWS CodePipeline
- AWS CodeBuild
- Cirrus CI
- Travis CI
- Teamcity (partial)

Containers

Images

- Select base image (FROM) built for **arm64**
- Registries will use a tag like “ARM 64” with e.g. OS/ARCH: “linux/**arm64**/v8”
- Multi-architecture container images based on manifest files are a recommended approach

Compatibility

- Can’t run images built for a different architecture (in this case, an x86_64 image on an arm64 host)

```
$ docker run --platform linux/amd64 nginx:1.23.3
…
exec /docker-entrypoint.sh: exec format error
```

- Emulation can help, but introduces performance and reliability issues

Software

- Most tooling supports **arm64** (registries, container runtimes, container orchestrators). Check tool’s documentation
- Software layer must be compatible with **arm64** (e.g. installing packages, building code, running applications)

AWS Graviton

Overview

- Designed by AWS to deliver the best price-performance for workloads running in Amazon EC2 (+ supported services)
- **General-purpose Arm64 processor** (3 generations from 2018-2023) which supports a wide variety of workloads
- Workloads include: web services, databases, caches, big data, analytics, encoding, gaming, HPC, ML, and blockchain

Tips

- Graviton has no hyper-threading / SMT. Each vCPU is a physical core, enabling performance at a low cost
- Lower cost and energy efficiency are fixed, but performance needs to be determined per workload
- Identify instances by **lowercase g letter:** C7**g**, M6**g**, R6**g**d, Im4**g**n, G5**g**, etc. Select AMIs available for **64-bit Arm**

Notes

- Instances benefit from the AWS Nitro System
- Graviton 1 A1 instances are legacy. Use Graviton 2 (e.g. M6**g**) or Graviton 3 (e.g. M7**g**)
- Some EC2 features such as Hibernate might not be available. Check service docs for latest updates