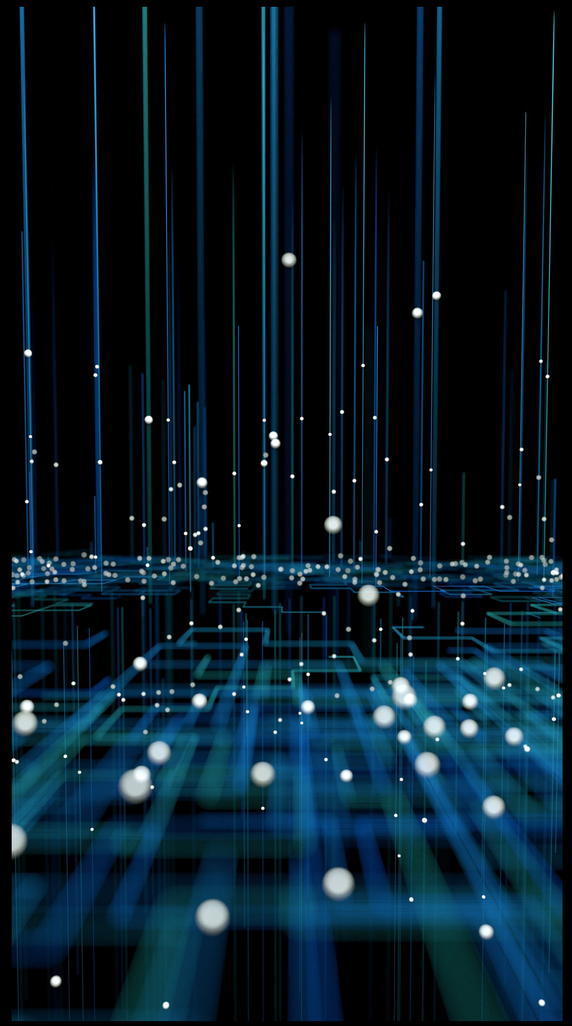
WHITE PAPER

The Insider's Guide to Building a Multi-Arch Infrastructure

A White Paper By Cheryl Hung,
Senior Director, Infrastructure Ecosystem, Arm

arm



The Insider's Guide to Building a Multi-Arch Infrastructure

ABSTRACT

Multi-architecture infrastructures let workloads run on the best hardware for the task (x86- or Arm-based) and can optimize price/performance ratios while boosting design flexibility, but migrating from a single- to a multi-arch framework can be tricky. Here's how early adopters are simplifying the process.

Is Migrating Worth the Effort?

One of the first things to evaluate, when considering adding a second hardware architecture to your development infrastructure, is what you're likely to gain from migration. After all, expanding the infrastructure is a major undertaking, since it has implications for just about every aspect of development, from the Infrastructure as Code setup and the CI/CD pipeline to packaging, binaries, images, Kubernetes upgrades, testing, scheduling, rollout, reproducible builds, performance testing, and more.

So why do it? Two reasons. First, a multi-arch infrastructure can reduce operating costs, and second, it can increase choice when making development decisions.

1. Better Price/Performance Ratios

Controlling the cost of cloud computing remains a challenge. Prices can change quickly and unexpectedly, suddenly making it less affordable to run the same workloads. Making changes to a workload can trigger unanticipated charges, and even small inefficiencies can add to infrastructure costs over time. When working to minimize the infrastructure budget, optimizing the price/performance ratio of hardware operations becomes an important tool for generating savings.

The Arm architecture, which is recognized for its low-power operation and efficient processing, can help reduce costs by making hardware processing more affordable. In side-by-side comparisons, RISC-based Arm architectures have consistently shown they work more efficiently than their CISC-based x86 counterparts. As a result, developers have found that running some or all of their workloads in the Arm architecture is an effective way to increase efficiency, and thereby optimize price/performance ratios.

> "Saving 40% on the EC2 instance bill for this service [after migration] is well worth the investment."
>
> – Liz Fong Jones, Field CTO, Honeycomb.io

2 . More Choice, More Flexibility

For quite a while, the x86 architecture dominated cloud-based offerings, but that's hardly the case anymore. Since 2018, when Amazon Web Service (AWS) launched their first 64-bit Arm-based CPU, Graviton, on the Elastic Compute Cloud (EC2), the ecosystem for Arm-based development has continued to expand. Other Cloud Service Providers (CSPs) have moved quickly to add their own Arm-based offerings, and Independent Software Vendors (ISVs) continue to introduce new cloud-based tools for Arm development. Today, the elements needed for Arm-based development are solidly in place, making it much easier for developers to find the cloud-native resources they need to run on Arm-based hardware.

At the same time, the ongoing shift toward Arm-based architectures, in mobile, automotive, and other areas, including laptops, has had an impact on cloud-based development, too. The widespread adoption of the Arm-based Raspberry Pi development platform, and the fact that the Arm architecture can be licensed for a wide range of use cases, from edge devices to data centers, has changed how many developers approach their designs. Not only are today's developers more familiar with Arm-based architectures, they have a wider range of choices when it comes to evaluating features and functionality. In some cases, developers are choosing not to work in legacy x86 environments because it can be painful to compile and build on their day-to-day developer machines.

Also, in a growing number of situations, the latest options may only be available for Arm-based architectures. Migrating to a multi-arch infrastructure makes it easier to take advantage of those choices, because workloads can run on the hardware that best matches the operating requirements.

> "48 of the top 50 Amazon EC2 customers use AWS Graviton processors for their workloads."
>
> – Danila Poccia, Chief Evangelist (EMEA), AWS, August 2022

These two trends – the rising cost of cloud computing and the increasing presence of Arm-based hardware architectures in a wider variety of applications – are driving the adoption of multi-architecture infrastructures. With a multi-arch infrastructure in place, workloads can run on x86 or Arm hardware without forcing developers to worry about the underlying architecture, so it's easier and faster to create, introduce, and maintain new features. Development teams are finding that the upfront work of migration is worth it, since it saves time and effort down the road.

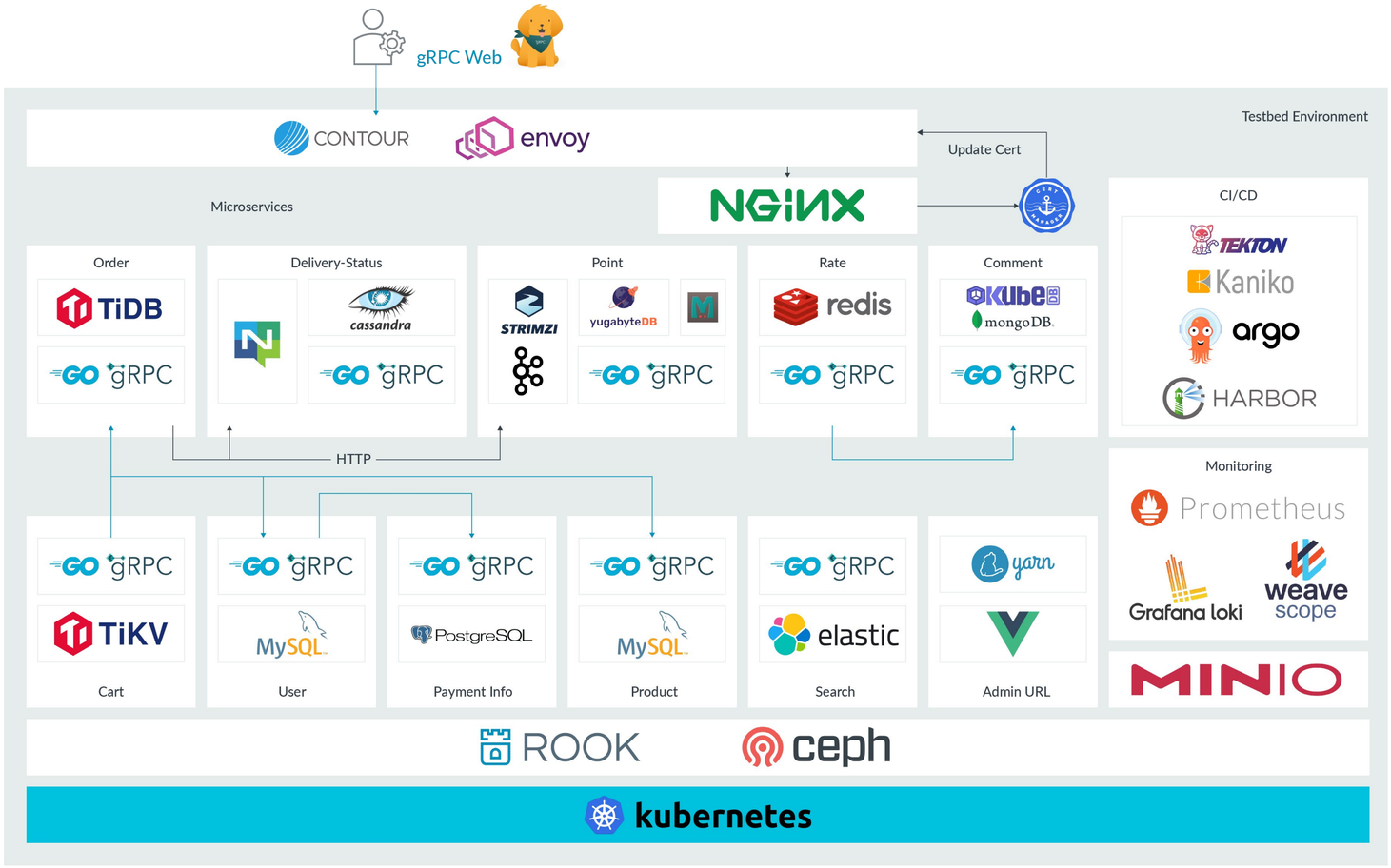


FIG. 1

Multi-Architecture Migration Touches Just About Everything

A Framework for Migration

From a planning standpoint, one of the biggest challenges for multi-arch migration is that it's not just one thing – it's pretty much everything. Consider the infrastructure for a sample e-commerce website, shown on the left.

The Kubernetes platform, for container orchestration, is a foundational element, but you also have a CI/CD pipeline, monitoring functions, and a series of container-based microservices. The setup is also likely to include storage of some kind, with or without statements attached.

Upgrading the Kubernetes instance is likely to be an important step of any migration, but it's only part of what needs to be considered. There are a number of other things to think about, such as the details of your Infrastructure as Code (IaS) setup, the nature of your CI/CD pipeline, and your process for creating reproducible builds. What do your packaging, binaries, images, and registries look like? What are your processes for testing, scheduling, rollouts, and performance evaluations? Also, you may have already optimized your setup to create a good balance between price and performance. Will migration, and the fact that you're touching so many elements of the setup, change that balance?

The actual mechanics of how you implement your multi-arch infrastructure will obviously depend on your individual setup. Some migrations will be more involved than others. But there are some general guidelines we can recommend, based on our own experience and what we've learned from early adopters.

FIG. 2

The Three Stages of Multi-Architecture Migration

Building on Best Practices

Our recommendations build on two basic assumptions: you're already using a cloud-native infrastructure, and you're running it on a public cloud. We also borrow from one of the best in the industry – the FinOps Project of the Linux Foundation. We like their approach to cloud-based management, and like how they divide complex operational tasks into three stages – inform, optimize, and operate.

For our purposes, the inform step is where you identify what you're running. The next step, optimize, is where you pick a subset of workloads to experiment with, and the third step, operate, is where you actually do all the necessary upgrades and begin deployment.

Let's take a closer look.

1. Inform

Make a list of everything involved in your existing development infrastructure. Start with a complete inventory of your entire software stack. What operating systems do you use and what images are you running? What resources do they rely on? Which libraries do you access, and what frameworks do you use to build, deploy, and test? How do you monitor or manage key aspects of operation, such as security?

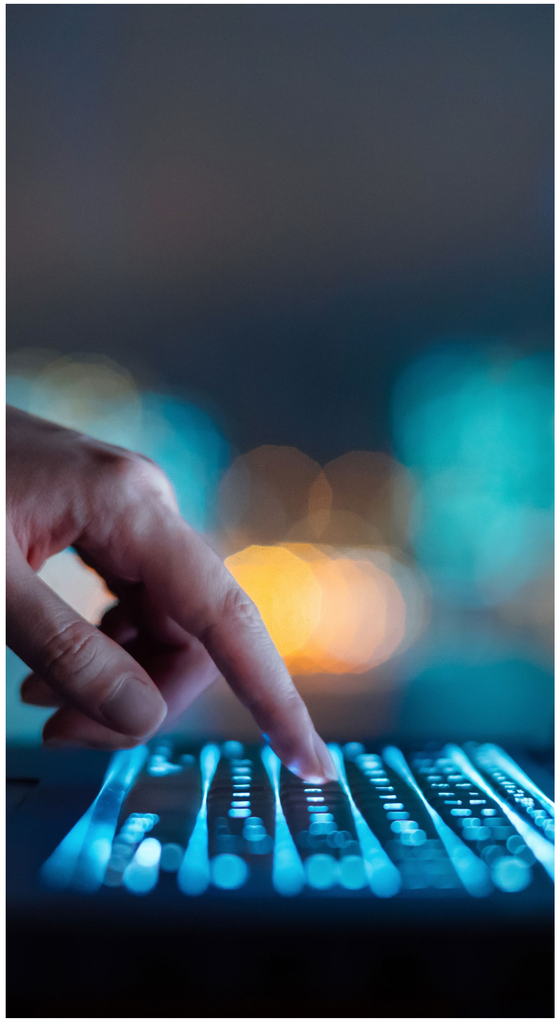
Capture everything – your list is likely to be quite long – and check that every item has Arm support. When verifying Arm support, keep in mind that different sources have different names for the 64-bit extensions of Arm. For example, the GNU Compiler Collection (GCC) categorizes them as AArch64, but the Linux kernel lists them under arm64.

Having created your inventory, check for hotspots. What are your most expensive compute items? Where do you spend the most in terms of your existing setup? Knowing where you'll find the highest proportion of executed instructions can help you understand which aspects of migration will offer the greatest opportunities for savings.

2. Optimize

Choose a few workloads and provision them to a test Arm environment. To keep things manageable, start small. Update a few container images and test syntax, check your performance tests, and modify your CI/CD pipeline to enable reproducible Arm builds.

Spin everything up to your public cloud and begin working on all the minor upgrades, changes, and "if statement" additions needed to run an additional architecture. There's no need to upgrade the entire environment before migrating an individual workload. For example, you can make changes without modifying the Kubernetes platform.



3. Operate

This is where it gets interesting, because you'll be building your Kubernetes cluster, revising the infrastructure, and rolling out your new processes.

The first decision is likely to be how you'll migrate your control plane and worker nodes. You probably don't want to move everything at once, and you'll want to think through the options. You might, for example, move control nodes first and work nodes second, or you might move them together, in small batches. The trade-offs for creating each cluster will reflect your software stack, node availability, and the nature of your workloads.

You'll also want to go through your scripts for cluster creation, adding in the changes for each hardware architecture. Having a mixture of x86 and Arm scripts will affect anything running in a DaemonSet controller. Make sure the correct image is pulled down for the architecture you're using.

With your new Kubernetes clusters in place, you're ready to start deploying. We recommend starting with a small subset, as part of a canary deployment, or running the new release candidates alongside the active production environment, in a blue/green deployment. Either way, it's best to start gradually, monitoring as you go.

When checking if scheduling is working as planned, the Kubernetes concept of node affinity can help streamline the setup. Using a combination of taints and tolerations, you can ensure that the right workloads are running on the right nodes. You may also want to adjust the number of requests for each architecture, finetuning the limits to optimize performance.

A Good Way to Experiment

Looking over these three stages of migration, you probably noticed that what you're doing in the first two stages – inform and optimize – is basic prototyping. That is, you're identifying functionality and using a small set of workloads as a proof of concept, to see if your test environment yields good results. It shouldn't take more than a few spare afternoons to complete these first two stages. As an experiment, it might even be worth doing the first two stages on their own, even if you're not sure you want to proceed with a full migration.

Two Real-World Examples

When looking for case studies of multi-arch migration, the easiest place to look is AWS Graviton, since that's the Arm-based architecture that's been running in the cloud the longest. Our first migration example, FusionAuth, continues to run a mix of architectures today, and the second, Honeycomb.io, ended up using their multi-architecture migration as a way to transition fully to Arm-based operation.

1. Case Study: FusionAuth

With over 10 million downloads, FusionAuth is one of the world's leading suppliers of identity and user-management solutions. They were one of the first to provide a developer-focused API for authentication and authorization, and that's the functionality their migration focused on.

The migration was prompted by a community member, who wanted to experiment with authorization on a Raspberry Pi development board. FusionAuth is a Java shop, which meant they had to find a Java Virtual Machine (JVM) that supported Arm (Java 17 was the first version to do so). FusionAuth added Java 17/Arm support to the code, and then updated Docker to target the Arm architecture with jlink and multi-arch builds. Because they run on Java, the FusionAuth lift for the migration to Arm was relatively small. The biggest hurdle was finding a JVM built for Arm. After that, it was a matter of working out any remaining kinks.

When FusionAuth's SaaS solution was in development (2019), the number of public-cloud regions running Arm was still fairly limited, so they had to be selective. After load testing FusionAuth using AWS Graviton servers and updating the JVM, they began official support for Arm in June of 2022. The expansion was quick, and as of March 2023, more than 70% of their SaaS server instances were running Arm.

For their load tests, FusionAuth chose to test login request, since password hashing makes logins an especially CPU-intensive process. Having tested 50,000 logins in the AWS EC2 environment, they found the Arm architecture handled between 26% and 49% more logins per second, and cost between 8% and 10% less than the equivalent setup running on an x86 architecture.

> **"I just switched a FusionAuth instance to arm64 and the transition was so smooth I couldn't even tell whether it's actually running the arm64 version."**
>
> – Hendy Irawan, Dunia Anak Alam Foundation CIO and FusionAuth User

2. Case Study: Honeycomb.io

With their observability platform, Honeycomb.io helps engineering teams gain a deeper understanding of their own production systems and their end-user experiences. They began experimenting with AWS Graviton in March 2020. Within a year, they had their first workloads in production. Just six months later, they had nearly all (92%) of their workloads and environments on Arm vCPUs. By April 2022, they were able to turn off the last of their x86 EC2 instances, and were running 99% on the AWS Lambda instruction set.

Honeycomb.io thought carefully about the order of their migration. They began with ingest workers because they were stateless, performance-critical, and relatively easy to scale out horizontally. Also, the ingest workers were written in Golang, so compiling for Arm was easy.

They prepared multiple environments at once and used a dogfooding environment to test their new deployments on themselves, as their end users would. They didn't use Kubernetes or container orchestration at first, choosing to use Terraform and Chef instead, so they could switch Arm Amazon Machine Images (AMIs) by enumerating all the dependencies for updates.

After the intake workers, they migrated their own workloads, in Apache Kafka, since they were the easiest to recompile and represented the highest compute spend. From there, they went down the list, migrating ad-hoc and one-off services and saving the toughest workloads for last. In reviewing their work and evaluating their roadmap, they decided to discontinue their use of x86 hardware. The decision to switch entirely to Arm reduced complexity, lowered operating cost, and generated a significant competitive advantage.

> "With AWS Graviton, we can scale up our product without increased operational toil, spend less on compute, and have a smaller environmental footprint"
>
> – Ian Smith, Engineering Manager, Honeycomb.io

DOGFOODING ARM ON ARM

When it comes to case studies for migrating infrastructures, it's worth noting that our own development teams, at Arm, went through the process and transferred our most compute-intensive operations to Arm in the cloud. (We wouldn't be a very good tech company if we didn't use our own products!)

In 2019, we migrated our on-premises, x86-based infrastructure for Electronic Design Automation (EDA), which we use to design, model, simulate, test, and analyze our circuit designs, to AWS Graviton. In the process, we improved performance by more than 60%, cut our costs in half, and generated a savings of more than 1 megawatt of power per day.

Turning Promise into Reality

At a high level, the goal of running a multi-architecture infrastructure is to have workloads run on the best hardware for their price/performance needs, without developers being concerned with the underlying architecture. But that doesn't mean it's an easy goal to obtain. To simplify the migration process, we recommend following a FinOps approach that divides tasks into three stages, beginning with a detailed inventory, doing a small test run, then proceeding to full updates and a monitored rollout.

Take the Next Step

To learn more from the experts about how to get started with multi-arch migration, find more resources on www.arm.com.