How AWS Graviton helps Independent Software Vendors accelerate growth and improve their margins on AWS

AWS Graviton2 for Independent Software Vendors

AWS Graviton2 for Independent Software Vendors: How AWS Graviton helps Independent Software Vendors accelerate growth and improve their margins on AWS

Table of Contents

AWS Graviton2 for ISVs

Publication Date: **January 6, 2022**

Abstract

Amazon Web Services (AWS) provides a broad and deep choice of Amazon Elastic Compute Cloud (Amazon EC2) instances to match the wide spectrum of computing needs of our customers such general purpose, compute optimized, memory optimized, storage optimized, and accelerated computing workloads. This enables customers to choose the most cost-effective instance type suitable for their particular workload. These instances are often based on the Intel architecture. AWS recently introduced a new family of instance types based on the ARM architecture – AWS Graviton2. These instance types provide up to 40% price performance improvement over comparable Intel-based instances. For Independent Software Vendors (ISV), this translates to lowering the Cost of goods sold (COGS) and improving margins.

This whitepaper provides a roadmap to help ISVs evaluate the suitability of Graviton 2 to their workload and a checklist to help build a project plan for adoption for the most common ISV use cases – Software-as-a-Service (SaaS), Marketplace AMI, and Direct. This paper also provides instructions, resources and best practices for each step in the migration journey.

Introduction

ISVs run virtually any cloud workload and compute use case on AWS including general purpose, compute optimized, memory optimized, storage optimized, and accelerated computing workloads. AWS provides a portfolio of purpose-built compute instances covering 275 instance types across 7 categories to serve the needs of our customers. With the rising popularity of the ARM architecture for server workloads, AWS introduced the first EC2 instances powered by the AWS Graviton Processor in 2018 followed by AWS Graviton2 Processor in 2019. (IDC estimates worldwide revenues for ARM-based servers grew 430.5% year over year in Q3/2020. For more information, see Worldwide Server Market Revenue Growth.) Graviton instances provide up to 40% better price performance than comparable Intel-based instances. For ISVs used to deploying on Intel-based hardware, this means transitioning their software to ARM in order to tap into the price performance benefits and cost savings potential of AWS Graviton.

This paper provides a checklist for software architects and developers working in the ISV sector who want to adopt AWS Graviton2 to accelerate growth and optimize margin. Whether you distribute your software offering via AWS Marketplace AMIs and container images, run a SaaS service on AWS, or distribute your software directly to customers. In each scenario, AWS Graviton2 instances offer the best price performance in their respective Amazon EC2 instance family. You can keep or reinvest cost savings and demonstrate higher margins to investors (important for ISVs with a SaaS model).

This paper also offers key considerations you should make when planning your transition to AWS Graviton2 and provides examples for popular tools, languages, and runtimes with 64-bit ARM support. AWS Graviton2 is one of the first implementations of the 64-bit ARM architecture for servers that meet the performance expectations of server workloads. ARM processors power most end user computing devices including mobile phones today. They offer higher power efficiency than x86 processors, making them cheaper to operate.

Background

This section provides background information on AWS Graviton2 and the benefits of a 64-bit ARM architecture over traditional software architectures. This information will help you assess whether AWS Graviton2 is a good fit for your application or service before moving on to a discussion of benefits and transition planning.

For years, AWS has been designing custom chips that enable faster innovation, deliver increased security, increase performance by offloading virtual functions, and reduce cost for our customers. This has led to innovations in terms of how customers secure workloads (Nitro Security Chip and Nitro Enclaves), benefit from enhanced throughput and latency for networking and storage I/O (Nitro Card with high IOPS EBS storage and up to 100Gbps networking) and virtualization technology (Nitro Hypervisor). The Nitro System protects hardware resources, improves monitoring and security posture, and benefits from better memory and CPU allocation for bare metal-like performance. All in a purpose-built, modular system.

AWS Graviton2 processors continue the tradition of silicon innovation and are custom-built by Amazon Web Services using 64-bit ARM Neoverse cores. These processors are optimized across AWS to deliver the best price-performance for cloud workloads running in Amazon EC2. AWS Graviton2 processors provide even more choice to help customers optimize performance and cost for their workloads.

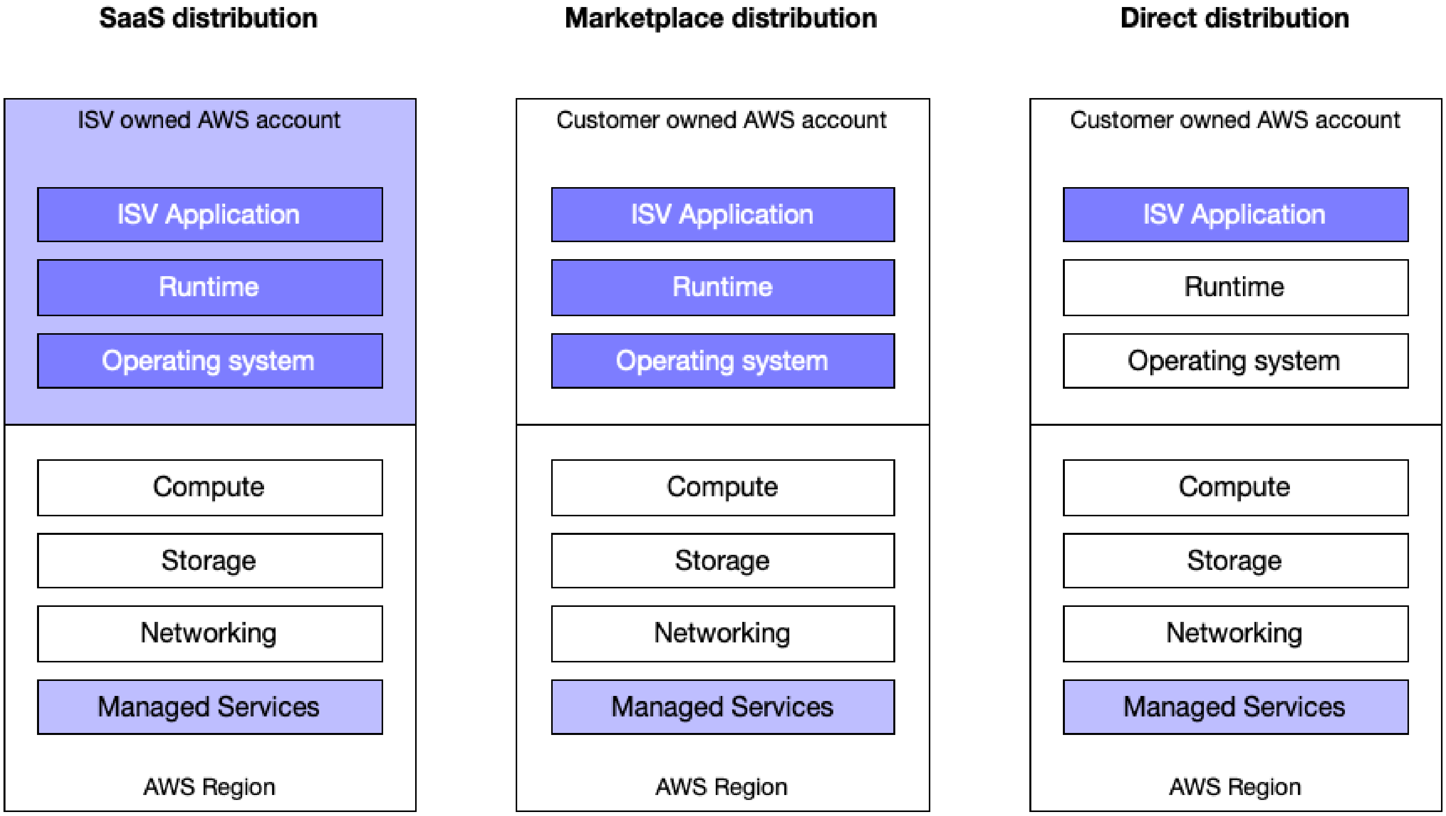
AWS Graviton2 processors power different instance types. At the time of publication, these are the M6g, C6g, R6g, and T4g instance types. The M6g instance type powers general purpose workloads such as such as application servers, microservices, gaming servers, mid-size data stores, and caching fleets. The C6g instance type is optimized for compute-bound applications that benefit from high performance such as high-performance computing (HPC), batch processing, media encoding, and CPU-based machine learning (ML). The R6g instance type offers a higher memory footprint for applications that process large data sets in memory including databases, in-memory caches, and real-time big data analytics. The T4g instance type is ideal for low-cost, burstable general-purpose workloads. For more guidance on benchmarking and optimizing your application or service on AWS Graviton2 instances, see Optimizing for Performance.

For more information about the benefits of AWS Graviton2 and the Amazon EC2 instances powered by AWS Graviton processors, see AWS Graviton.

Use case scenarios

This section introduces four use cases for modernizing ISV applications and transitioning them to AWS Graviton2. It also provides a short summary of each use case and outlines the benefits of AWS Graviton2 before covering transition activities in the next section. Select the use case that is closest to your situation to get started.

The benefits of using AWS Graviton-based instances for ISV applications differ slightly based on how you distribute your software to end customers. Figure 1 illustrates the differences and commonalities between three primary use cases (distribution model of your service or application) and highlights service dependencies as another important consideration (fourth use case). It calls attention to AWS account ownership, control over application source code, runtime, operating system, as well as service dependencies the application requires to function. The components you should consider during a transition to Graviton2 depend on how you distribute your software to your customers.



Software distribution models and service dependencies

In the following sections, we describe in more detail each software distribution model and benefits you can expect to derive from transitioning to AWS Graviton2.

SaaS distribution (as SaaS-based product or subscription)

The SaaS distribution model offers the greatest degree of flexibility for ISVs to distribute their software. The ISV has full control over AWS account ownership, application deployment, runtime, operating system, and compute resources. Customers procure subscriptions through AWS Marketplace (SaaS-based product) or through the ISV directly.

Benefits of AWS Graviton2 in the SaaS distribution model:

- Up to 40% better price performance compared to current generation Intel instances in your production and trial stacks.
- Growing ecosystem of open-source and commercial vendor support for the 64-bit ARM architecture.
- Increased reliability by tapping into an additional capacity pool for your production deployments.
- Develop, build, and test 64-bit ARM versions for your software on AWS Graviton2 using popular tools such as Travis CI with AWS Graviton support.

Marketplace distribution (as AMI-based or Container-based product)

The Marketplace distribution model enables ISVs to offer Amazon Machine Images (AMI-based) or Container-based products to end customers. Customers can procure a license or subscription through the AWS Marketplace and deploy the application in their AWS account. The ISV dictates the fulfilment options of the offer such as the operating systems and EC2 instance types supported by the application.

Benefits of AWS Graviton2 in the Marketplace distribution model:

- Extend your market reach by offering a 64-bit ARM architecture option in addition to your existing x86 offering (this includes Marketplace AMIs or packages and software agents installed on Linux servers or virtual machines).

- Attract new customers looking for better price performance when deploying your software on EC2 instances in their AWS account.
- Develop, build, and test 64-bit ARM versions for your software on AWS Graviton2 using popular tools such as Travis CI with AWS Graviton support.

Direct distribution (as application binary)

The direct distribution model requires ISVs to consider the deployment environment in which the ISV application runs. Customers download the application binary from the ISV’s website or via their operating system’s package manager. The ISV has to ensure that the application supports different combinations of operating systems, runtime, and platforms in order to reach a broad market.

Benefits of AWS Graviton2 in the Direct distribution model:

- Tap into the growing demand for commercial 64-bit ARM software for servers (this includes packages and software agents installed on Linux servers or virtual machines).
- Develop, build, and test 64-bit ARM versions for your software on AWS Graviton2 using popular tools such as Travis CI with AWS Graviton support.

Service dependencies

Service dependencies include open-source workloads such as web servers, caching fleets, container clusters, and open-source databases on which your application relies to properly function. Depending on your distribution model, you either manage these dependencies in your environment or rely on customers to deploy them alongside your application in their target AWS account.

Benefits of AWS Graviton2 for service dependencies:

- Improve price performance of common infrastructure components such as open-source databases, container clusters, and caching fleets by running them on AWS Graviton-based instances.
- Eliminate undifferentiated heavy-lifting by transitioning EC2-based dependencies to a managed service model using services such as the Amazon Relational Database Service (Amazon RDS), Amazon Elastic Container Service (Amazon ECS), Amazon Elastic Kubernetes Service (Amazon EKS), and Amazon ElastiCache that support AWS Graviton-based instances today.

Planning checklist

This section provides an overview of transition activities before moving on to a detailed discussion of each activity and its associated technical and architecture considerations in subsequent sections. Use this section as a reference to plan your approach.

Table 1 introduces transition activities as a *planning checklist*. This checklist is based on the definition of software distribution models introduced in the previous section. It enables you to identify relevant transition activities based on which of the distribution models most closely resembles the way you distribute software to customers. You can use this checklist as a reference when planning and executing your transition to identify constraints and opportunities for quick wins. For example, you may be able to gain experience with Graviton2 by configuring your build environments to build and test your application on Graviton2. You may also start by transitioning service dependencies such as open-source databases or caching servers before transitioning your application.

Table 1 – Transition Activities Planning Checklist

Model	Plan	Transition	Optimize	Review
SaaS	Identify and resolve all software dependencies Pay attention to architecture dependent and proprietary commercial software Consider control plane and data plane artifacts	Build and test source code on 64-bit ARM Update build scripts to support 64-bit ARM Build multi-architecture AMIs and container images Update container registry	Baseline performance on x86 and Graviton2 instances Upgrade host operating system and container runtime Tune and optimize compiler settings	Budgeting and approval for test fleets Reservations and other cost optimization strategies

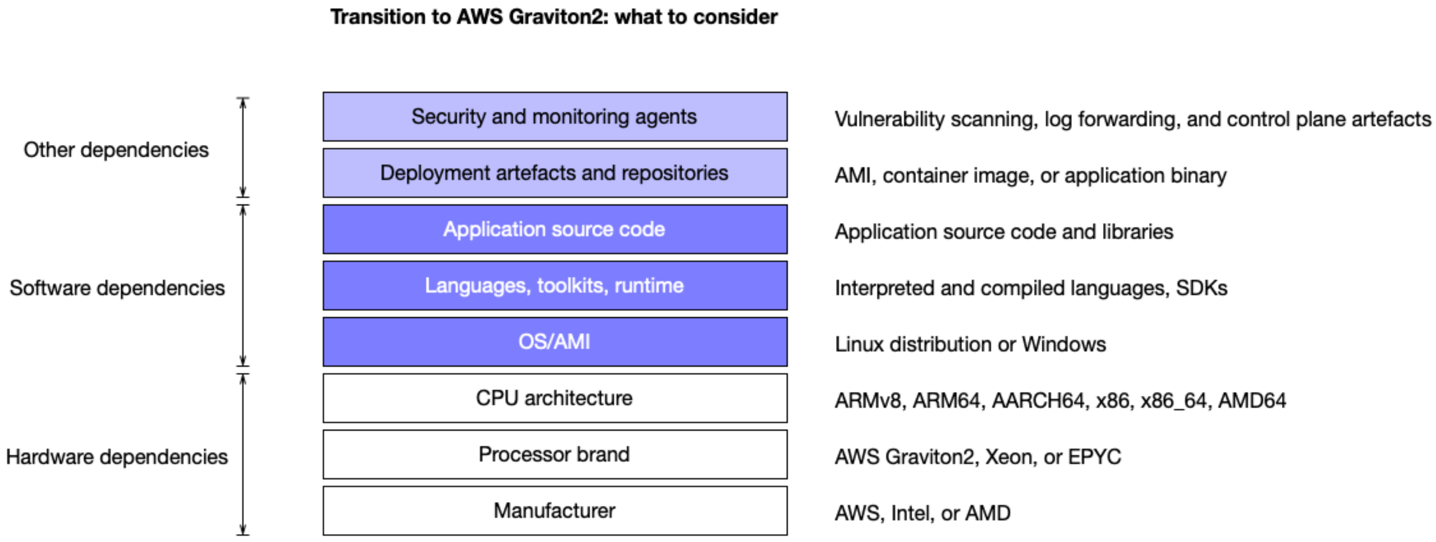
Model	Plan	Transition	Optimize	Review
	Consider data plane artifacts only if using managed services	Use deployment procedures such as canary deployments or blue/ green cutover	Benchmark performance on different instance sizes	
Marketplace	Identify and resolve all software dependencies Pay attention to architecture dependent and proprietary commercial software Packaging of Marketplace AMI or container images	Build and test source code on 64-bit ARM Update build scripts to support 64-bit ARM in addition to x86 Build multi-architecture AMIs and container images Update container registry	Baseline performance on x86 and Graviton2 instances Upgrade host operating system and container runtime Tune and optimize compiler settings Benchmark performance on different instance sizes	Budgeting and approval for test fleets

Model	Plan	Transition	Optimize	Review
Direct	Identify and resolve all software dependencies Pay attention to architect ure dependent and proprieta ry commercial software	Build and test source code on 64-bit ARM Update build scripts to support 64-bit ARM Installation and configuration guidelines	Baseline performance on x86 and Graviton2 instances Tune and optimize compiler settings	Budgeting and approval for test fleets
Dependencies	N/A if using managed services on AWS	Cut over managed service to 64-bit ARM in production	N/A if using managed services on AWS	Reservations and other cost optimization strategies

Planning your transition

This section will help you prepare your transition project and highlights things to consider. In most cases, your application or service consists of multiple components and their dependencies. Being aware of what these dependencies are will help you identify constraints and opportunities for quick wins. If you are primarily interested in the technical activities associated with transitioning to AWS Graviton2, see Transitioning Your Service or Application.

The following figure illustrates the typical components of an application stack. Your application may differ or you may not have full control over some components depending on your software distribution model. For example, you may have full control over all components of the stack if you are distribution software as a SaaS-based product. However, you may have less control over some aspects such as operating systems and software agents if you distribute your software directly to customers (and customers are responsible for deploying and configuring your software).


Transition to AWS Graviton2: what to consider
Other dependencies
Software dependencies
Hardware dependencies
Security and monitoring agents
Deployment artefacts and repositories
Application source code
Languages, toolkits, runtime
OS/AMI
CPU architecture
Processor brand
Manufacturer
Vulnerability scanning, log forwarding, and control plane artefacts
AMI, container image, or application binary
Application source code and libraries
Interpreted and compiled languages, SDKs
Linux distribution or Windows
ARMv8, ARM64, AARCH64, x86, x86_64, AMD64
AWS Graviton2, Xeon, or EPYC
AWS, Intel, or AMD

Typical components of your application stack

The following sections provide additional details around each application component and the specific issues you may encounter when transitioning to AWS Graviton2.

Operating system (OS)

Most popular Linux distributions offer out-of-the-box support for AWS Graviton2 include Amazon Linux 2, Red Hat Enterprise Linux, Ubuntu, and SUSE Linux Enterprise Server. Use the latest version of these operating systems to benefit from performance enhancements. Consider which operating systems and versions you or your customers are running in their production environment. Verify that the operating system offers 64-bit ARM support using the AWS console and CLI. Filter by owner, region, platform and architecture to check whether a combination is supported today. At the time of publication, Windows is not available for AWS Graviton2. For an up-to-date-list of supported operating systems, see the AWS Graviton Getting Started Guide.

Language and runtimes

Verify which languages and runtimes make up your codebase and whether your code dependencies include native language extensions (such as Java JNI's) or artifacts (such as x86 shared objects where equivalent ARM versions should be created). Interpreted and bytecode-compiled languages such as Python, Java, Node.js, and .NET Core on Linux mostly run without modifications. This means you should be able to run these applications on AWS Graviton2 by simply installing the corresponding runtime.

For example, the Java Virtual Machine (JVM) runtime added 64-bit ARM support in version 8 and above. (Java has been available in multiple forms for ARM for a long time. OpenJDK and Oracle JDK added support for 64-bit ARM in JDK8 and above. Servers and cloud workloads typically adopt LTS versions of Java which include JDK8, 11, and 15 which all have 64-bit ARM support and later versions provide more 64-bit ARM specific optimizations for best performance.) Check the Getting Started Guide to identify exceptions (such as platform-specific artifacts) that apply to your language and how to work around them. Applications written in compiled languages such as C/C ++ and Go will need to be recompiled these languages provide mature and optimized support for both 64-bit ARM and x86. For example, this includes the GCC and LLVM for C/C++ and the golang/go compiler for Go. (Some languages provide a built-in cross compiler (like Go), some do support multiple architecture in the sense that you can compile a compiler for a target but not a compiler for multiple targets at the same time.)

Container services

If your application is deployed using container services, check for multi-architecture support for containers and infrastructure dependencies. Containers are architecture specific and you will need to build new container images to support 64-bit ARM. Multi-architecture support helps simplify the consumption of container images across x86 and ARM. For example, Docker supports multi-architecture images transparently via the Buildx tool and the majority of Docker official images are now multi-architecture enabled.

You can deploy multi-architecture container images in Amazon Elastic Container Service (Amazon ECS) and Amazon Elastic Kubernetes Service (Amazon EKS). This enables you to optimize your clusters by offering support for both x86 and Graviton2 hosts. Other considerations include specific infrastructure dependencies such as proxies or service mesh components. For more information about the status of open-source projects and their 64-bit ARM and Graviton2 support, see the Getting Started Guide.

Software agents

If you distribute your application as SaaS, work with your operations team to identify your dependencies on software agents. Most security, monitoring, and logging use cases require operations teams to install software agents on virtual machines and container hosts. Lack of support for 64-bit ARM can be a blocker if your security and application performance monitoring processes rely on these agents. Most first-party software agents on the AWS platform provide 64-bit ARM support. This includes the Amazon CloudWatch, Amazon Inspector, and AWS Systems Manager agents. A growing ecosystem of third-party vendors offer a 64-bit ARM version of their software agents. Examples include security solutions (CrowdStrike, Qualys, Rapid7, Snyk, and Tenable) and observability and monitoring solutions (Splunk, New Relic, Datadog, and Honeycomb).

Build systems

Consider the state of multi-architecture support for continuous integration/continuous delivery (CI/CD) pipelines and build systems. Maintaining multiple versions of compilers, libraries, and support binaries for different architectures has traditionally been a cumbersome, time-consuming, and an error-prone process. Modern CI/CD tools offer facilities to build and test code submitted to code repositories with minimal manual intervention.

First party CI/CD tools on the AWS platform such as AWS CodeCommit, AWS CodeBuild, AWS CodePipeline, and AWS CodeDeploy provide out-of-the box support for 64-bit ARM. There is a growing ecosystem of third-party CI/CD vendors that are adding 64-bit ARM support to their software offerings. These include Jenkins, GitLab, CircleCI, and TravisCI. For example, GitLab now offers first class support for AWS Graviton and the ARM architecture in its CI/CD tooling.

Edge cases

You may encounter edge cases in older, monolithic applications. While these are rare, they may impact on your ability to transition to AWS Graviton2. Examples for edge cases include proprietary legacy applications, hand-tuned code written for x86 in low-level languages, or workloads that depend on specific features of the x86 architecture. If this is the case, consider whether AMD-backed instance types such as the M5a, C5a, and R5a are a better choice for your application. While AMD-backed instance types do not provide the same price performance advantages compared to AWS Graviton2, they offer a seamless transition and potential cost savings.

Transitioning your service or application

This section provides more details regarding the individual steps involved in transitioning an application to Graviton2. In the interest of simplicity, we assume this application is written in a bytecode-compiled language (e.g., Java or .NET Core) and deployed as one or more virtual machines in AWS. Some of the concepts described in the following may differ when the application is deployed via a different method (such as container services) or distributed to users to be run in their own cloud environments (AWS Marketplace AMIs). However, the high-level sequence of steps will not change.

Resolve code dependencies

Start with code dependencies such as software libraries and determine whether you are able to run them in a 64-bit ARM environment. This includes questions such as how current is your code base and which version of languages and runtime do you currently use. Work with your engineering team to identify code ownership (which can often be a problem for older codebases) and figure out external dependencies (with teams inside your company as well as any dependencies on third parties). For example, Java virtual machines (VMs) typically support ARM64 from JDK8 versions onward. Application code and packages that were developed and built against older versions may need to be upgraded first. Do not forget other components such as Software Development Kits (SDKs), OS agents, and control plane artifacts in this inventory. This inventory will help you identify which parts of your application are easier to transition.

Upgrade operating systems

Upgrade your operating system to a version that supports the Graviton2 processor and offer the full performance entitlement. Your production environment may be running an outdated version of your operating system. This means you may not have access to the latest drivers and platform-specific fixes and optimizations. Build and test your software on the newest x86 version of your operating system before attempting a transition to 64-bit ARM. For example, Amazon Linux AMI is nearing its end-of-life date and no 64-bit ARM support is available for this version of Amazon Linux. Upgrade to a higher version of Amazon Linux 2 and test your application on an Intel or AMD-backed instance first (e.g., M5 or M5a). Then transition your application to AWS Graviton2 (M6g) to retest. This ensures that older versions of your operating system do not become a bottleneck.

Upgrade language runtimes

Perform a runtime upgrade to the most recent version of your programming language to get the best performance. For example, JDK11 and higher offers many optimizations for 64-bit ARM. Consider upgrading to the latest stable JDK release to gain access to these optimizations when running your code on Graviton2. Additionally, .NET 5 has significantly improved performance by adding 64-bit ARM-specific optimizations in .NET libraries, providing you with access to the full performance entitlement on Graviton2.

While prior 64-bit ARM runtime versions for your programming language may exist, they do not always offer optimal performance. For example, consider the changes in garbage collection (GC) algorithms between Java versions and their impact on variables such as throughput. Upgrading to a higher version of your programming language may require an incremental process of refactoring and testing. Older application code may rely on interfaces and methods that are deprecated by a higher version of the programming language. If an upgrade to a higher version is not an option, then prior versions of your runtime can serve as a *temporary* stop gap solution for test and evaluation purposes.

Transition codebase and packages

Review your codebase and begin the transition to Graviton2. This includes your application code and any other software dependencies defined in metadata files, container manifests, or Infrastructure-as-Code templates. Start by upgrading old code dependencies to new versions. For example, you may need to upgrade Java libraries with native language extensions to their latest versions for your code to build and run on 64-bit ARM. Repeat this process until your code successfully builds and runs in your new target environment. Identify very old dependencies and decide whether you can retire them. This may require refactoring and retesting your code in case the dependency is not working as expected.

Test and benchmark your code

Test and benchmark your code on multiple architectures to verify that your build is correct. This often involves running shadow test fleets (of both 64-bit ARM and x86-based instance types). Update your deployment scripts and Infrastructure-as-Code template to account for different instance types, 64-bit ARM compatible AMIs, and other differences between your environments. Upgrade your build pipeline to build on 64-bit ARM and benchmark performance metrics (throughput, latency) of your code running on AWS Graviton2 against x86 builds. Consider stress testing your deployments to establish how your workload performs under peak and above peak loads. This ensures that your code is making effective use of the Graviton2 performance advantage. For more information about compiler flags in languages such as C/C++, see the Getting Started Guide.

Tune and test

Tune your software, test again, and repeat the process of iterative refactoring, testing, benchmarking, and tuning until all remaining dependencies have been completed. This might involve micro benchmarking and profiling your application code. You may also want to consider special optimizations for cryptographic or machine learning workloads. For more details on how to optimized for performance and details such as optimized compiler flags, see Optimizing for Performance.

Additional considerations

Update Amazon Machine Images (AMI) or container registries

Other considerations to make after you transition your service or application to 64-bit ARM include updating build scripts, machine or container images, and container registries. Whether you follow the SaaS model or distribute your software directly to customers, you must update your CI/CD pipelines to add support for building on 64-bit ARM machines. Other issues include building Amazon Machine Images (AMI) and Docker containers for the 64-bit ARM architecture in addition to your existing x86 artifacts. If you follow the Marketplace model, update your AWS Marketplace listings and container registries so customers find the right image for their use case.

Update deployment scripts

If you follow the SaaS distribution model, you also need to think about operational concerns as you transition your application or service to support 64-bit ARM. For example, this might mean updating your deployment scripts to weight in Graviton2 instances in parallel to your Intel or AMD instances. While out of the scope of this paper, consider using deployment strategies such as Canary Deployments or Blue/Green Deployments. For more information, see the Deployment Strategies section of the Introduction to DevOps on AWS white paper. AWS recommends you start with test and staging accounts before moving to production accounts, and to keep the changes limited to a small percentage of customers at a time.

Optimizing for performance

This section offers guidance for optimizing your source code to run on AWS Graviton2 instances. Due to the difference in CPU architecture, running code that is not optimized for the AWS Graviton2 processor may result in suboptimal performance. At the end of this section, you should be able to identify these differences and understand how to remediate them so you can make the most of AWS Graviton2.

Understand key advantages

One of the major differences between AWS Graviton2 instance types and other instance types is their vCPU to physical processor core mapping. Every vCPU on a Graviton2 processor is a physical core. This means there is no Simultaneous Multi-Threading (SMT) and more isolation between vCPUs. By contrast, every vCPU on a 5th generation instance type with Intel processor (such as M5, C5, and R5) is a hyper-thread. This means vCPUs share resources and there is less isolation than in the case of Graviton2.

Key advantages of the AWS Graviton2 processor:

- Feature sets optimized for cloud workloads, reducing the overheads of interrupts and context switching.
- Large L1 and L2 caches for every vCPU which means a large portion of your workload will fit in cache without having to go to memory.
- Every vCPU is a physical core, meaning more isolation between vCPUs and no resource sharing except the last level cache and memory system.
- Cores connected together in a mesh with ~2TB/s of bisection bandwidth, allowing applications to move very quickly from core to core when sharing data.
- No NUMA concerns, meaning every core sees the same latency to every other core and to DRAM.

Use optimized compiler flags

When targeting modern processors, the right compiler flags can lead to considerably better performance. New versions of compilers will begin emitting instructions specific to the CPU architecture by default. However, until they do, specific compiler flags allow you to enable new features to obtain higher performance for tasks handled by the CPU such as managing a mutex.

For example, AWS Graviton2 processors implement new atomic instructions (referred to as large-system extensions or LSE) to support scalable performance on larger instance sizes for applications implementing synchronization locks (e.g., databases). In this specific case using the outline-atomics flag will yield almost the same benefits, but results in code that is backward compatible (i.e., will run on older ARM CPUs like Graviton1-based Amazon EC2 A1 instances). For more information on using compiler flags, see the AWS Graviton Getting Started Guide.

Upgrade operating systems

Use the most recent 64-bit ARM release of your operating system whenever possible (such as Amazon Linux 2 and Ubuntu 20.04). The latest release of your operating system offers libraries that were built with optimized compiler flags (as previously mentioned, this can lead to considerable differences in performance). For example, the latest 64-bit ARM release for Amazon Linux 2 ships with libraries such as *libc* that are optimized for AWS Graviton2. This leads to better performance in more scenarios.

Tune low level code

Identify and tune any low-level code that includes architecture-specific CPU instructions. While uncommon in application programming, some source code or libraries may use highly optimized inline assembly code to achieve maximum performance from a particular CPU architecture. Due to different instruction sets implemented by different CPU architectures, source code that offers only one optimized implementation (such as x86) will not perform well on Graviton2 (which is ARM-based) so the application may fall back to a generic, slower implementation which means you will not see the full performance entitlement of AWS Graviton2 instances. Identifying such source code highlights performance critical routines that should also be implemented for AWS Graviton2.

Test performance on multiple instance sizes

When doing performance benchmarking, evaluate both ends of the instance size spectrum to detect performance bottlenecks that may occur only on the small or very large instance sizes of an instance family. For example, performance bottlenecks may only occur on the larger sizes of an instance family while your software performs well on the smaller instance sizes of the family. Benchmark multiple instance sizes in a systematic fashion to detect such bottlenecks and provide sizing guidance to your cloud operations team or end customers to help guide their instance size selection criteria.

Reviewing your cost structure

AWS provides a number of options for you to balance the need for instance flexibility and cost savings. This section summarizes common recommendations to help you gain visibility and manage cost. For a deep dive on cost optimization best practices, see the links provided in each of the subsections.

Cost metrics and workload management

Talk to your finance team about the need to run small test fleets during the testing and benchmarking stage. Look at flexible pricing options such as Spot Instances to reduce the cost of research and development. For more information, see the Leveraging Amazon EC2 Spot Instances at Scale white paper. Develop specific cost-related metrics that allow you to measure resource utilization, instances turned off daily, and instances to which cost tags have been applied. Other measures include workload management and the ability to turn off development, test, and staging workloads when not in use. This enables you to keep cost low while enabling development teams to perform test and benchmarking tasks on multiple instance types and architectures.

Instance fleets

Phasing in AWS Graviton-backed instances into your production fleets may require you to run multiple instance types and support different architectures. Consider optimizing your production fleets for instance type flexibility by making use of constructs such as Amazon EC2 Fleets and Compute Savings Plans. With Amazon EC2 Fleet, you provision capacity across EC2 instance types and across purchase models to achieve your desired scale, performance and cost. You can combine On-Demand and Spot purchasing options and specify an unlimited number of instance types. With Amazon EC2 Compute Savings Plans, you gain the greatest flexibility in terms of instance family, size, Availability Zone (AZ), region, operating system or tenancy at an up to 66% lower cost compared to On-Demand rates. Use Compute Savings Plans to maximize cost savings across parameters such as instance type, architecture, operating system, tenancy model, and region. Additional capabilities include multi-architecture support in Auto-Scaling Groups (ASG), which enables you to configure both Graviton2 and x86-based Amazon EC2 Instances in the same Auto-Scaling group with different AMIs.

Pricing model

Select the right pricing model for your workload as you ramp up your AWS Graviton usage. Amazon EC2 On Demand allows you to pay for compute capacity by the hour or the second (depending on which instances you run) without longer-term commitments or upfront payments. Amazon EC2 Spot Instances allow you to request spare Amazon EC2 computing capacity to run fault-tolerant (interruptible) workloads for up to 90% off the On-Demand price. Amazon EC2 Reserved Instances provide deep discounts (up to 72% lower compared to On Demand prices) when you commit to a 1- or 3-year term and instance type. Finally, Amazon Savings Plans offer significant savings over On Demand, just like EC2 Reserved Instances, in exchange for a commitment to use a specific amount of compute power. Compute Savings Plans automatically apply to EC2 instance usage regardless of instance family, size, AZ, region, OS, or tenancy.

Final considerations

Once you have successfully completed the transition of your application of service to AWS Graviton2, consider performing the following activities.

Review Ramp-up and go-to-market activities

Consider the ramp-up and go-to-market activities that go along with introducing 64-bit ARM support for your application. This includes activities around creating customer awareness and field enablement to make sure your field is in a position to respond to customer questions regarding AWS Graviton2 support. It may also include briefing systems integrators and consulting partners to ensure adequate capacity exists in the field to support customer projects. Briefing systems integrators and consulting partners primarily applies to the AWS Marketplace and Direct distribution models.

If you are providing a SaaS-based product, evaluate whether customers should have the option to choose the platform on which their instance of an application is deployed and whether you intend to pass on or reinvest the cost savings realized by running customer stacks on AWS Graviton2 instances.

Update installation and configuration guidelines

If you follow the AWS Marketplace or Direct distribution model, your customer is likely to expect guidance when deploying your software on AWS Graviton2. You may need to rewrite sections of your installation and configuration guidelines to cover issues such as sizing considerations and instance type selection. This may include highlighting differences between Graviton2-based instance and other EC2 instance types such as the absence of hyperthreading. It may also include guidance concerning the operating systems the 64-bit ARM version of your application supports and the AWS Regions in which customers can deploy it (as container image or Marketplace AMI).

Update product definitions in AWS Marketplace

If you are using AWS Marketplace to distribute your application as an AMI-based or Container-based product, you should plan to update product definitions. For example, this includes building 64-bit ARM AMIs that can be deployed onto Graviton2 instances in a customer account. This could also include packaging multi-architecture container images and updating your container registry so customers can find the right container image for their architecture. You may also need to update product metadata such as the supported regions, instance types, or container environments in which customers can deploy your application. For more information, guidelines, and documentation, see Submitting your Product.

Conclusion

This document provided you with a checklist to accelerate your transition to AWS Graviton2. It summarized the key reasons ISVs with SaaS-based, Marketplace-based, and direct distribution models make this transition. It highlighted benefits such as extending your market reach by offering 64-bit ARM versions of your software and benefiting from an improved price performance ratio when using Graviton2 to run your SaaS workloads. It discussed typical components of an application stack you should consider when planning this transition and offered guidance in the form of transition process and checklist for ISVs. It concluded by offering guidance for reviewing your cost structure and other considerations such as ramp-up and go-to-market activities for your offering.

The document covered the most common cases based on our experience working with ISVs. The ecosystem around 64-bit ARM support for popular operating systems, open-source technologies, and language or runtime is constantly evolving. Please reach out to your AWS account team if you need further guidance.

Contributors

Contributors to this document include:

- Karsten Ploesser, Senior ISV Solutions Architect, Amazon Web Services
- Jeff Underhill, Principal Compute GTM Specialist, AWS Business Development
- Arthur Petitpierre, Senior Specialist Solutions Architect, Amazon Web Services
- Csaba Csoma, Software Development Manager, Annapurna Labs
- Zi Shen Lim, EC2 Graviton Specialist, AWS Business Development
- Tyler Lynch, Senior ISV Solutions Architect, Amazon Web Services

Document revisions

To be notified about updates to this whitepaper, subscribe to the RSS feed.

Change	Description	Date
Minor updates	Edits for accuracy	January 6, 2022
Initial Publication	First Publication	March 2, 2021

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents current AWS product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.