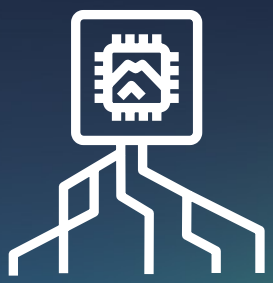
AWS Graviton Fast Start

A 4-step adoption plan for Amazon EC2



Introduction

AWS Graviton processors are custom designed by AWS to enable the best price performance for workloads in Amazon EC2. Amazon EC2 instances powered by AWS Graviton2 processors provide up to 40% better price performance over comparable fifth generation x86-based instances for a wide variety of workloads. The latest AWS Graviton3 processors add significant performance jumps over Graviton2, providing 25% better performance for a wide range of compute-intensive workloads. For a quick overview of Graviton processors and Amazon EC2 instances powered by Graviton, visit the AWS Graviton page.

We have seen many customers adopt Graviton with minimal effort and continue to enjoy the significant price-performance benefits they realized. This document outlines a framework to help you quickly move your own workload to Graviton-based EC2 instances with ease, all based on the best practices we've found from working with thousands of customers.

This plan is designed so a single engineer or small team can accomplish all four steps, with each step split into two subtasks.

We have outlined a 4-step plan in this document that is suitable for many applications, but based on the complexity of your application, some migrations take more time and others less. Regardless of the application complexity, the approach and high-level steps described here remain the same.

HOW TO IDENTIFY A GOOD TARGET WORKLOAD

A good candidate for Graviton adoption is a workload running on Linux or BSD, built either using open-source components or source code that you control. Having full access to the source code of every component allows you to make any necessary changes quickly and easily as part of this adoption plan. If you use third-party software, many ISVs already support the Arm64 architecture implemented by AWS Graviton processors. If you use third-party software that does not support Arm64, reach out to us on the re:Post community support forum.

If you get stuck at any step, feel free to reach out to Graviton experts on the re:Post community support forum. You can get started for free with Graviton-based instances by leveraging the T4g free trial that offers t4g. small instances free for up to 750 hours per month through December 31st, 2022. Check out the Amazon EC2 FAQs page for more details.

A good candidate for Graviton adoption is a workload running on Linux or BSD, built either using open-source components or source code that you control. Having full access to the source code of every component allows you to make any necessary changes quickly and easily as part of this adoption plan. If you use third-party software, many ISVs already support the Arm64 architecture implemented by AWS Graviton processors. If you use third-party software that does not support Arm64, reach out to us on the re:Post community support forum.

TOP TIP

Task 2 will require you to build the application/workload list of dependencies. If you need information from other teams, it might be useful to ask them for this information before you start the first task, so that you could have their answers by the time you reach Task 2.

Agenda

The following plan has been organized into a logical sequence of steps as follows:

- **Step 1: Learning and exploring**
  - Task 1 – Review key documentation and software support for Graviton
  - Task 2 – Explore your workload, and inventory your current software stack
- **Step 2: Plan your workload transition**
  - Task 3 – Install and configure your application environment
  - Task 4 – Build your application(s) and/or container images
- **Step 3: Test and optimize your workload**
  - Task 5 – Testing and optimizing your workload
  - Task 6 – Performance testing
- **Step 4: Infrastructure and deployment**
  - Task 7 – Update your infrastructure as code
  - Task 8 – Perform Canary or Blue-Green deployment

Step 1: Learning and exploring

TASK 1

Review key documentation and software support for Graviton

Start by watching re:Invent 2020 – Deep dive on AWS Graviton2 processor-powered EC2 instances and re:Invent 2021 – Deep dive into AWS Graviton3 and Amazon EC2 C7g instances, which will give you an overview of the Graviton-based instances and some insights on how to run applications depending on their operating system, languages, and runtimes.

Keep learning by watching re:Invent 2021 – The journey of silicon innovation at AWS to better understand Amazon's commitment to pushing the envelope with custom silicon.

Finally, spend some time reading relevant sections of the Getting started with AWS Graviton repository, which will act as a useful reference throughout your adoption.

TASK 2

Explore your workload, and inventory your current software stack

To start the migration, the first thing you need to do is inventory your current software stack so you can identify the path to equivalent software versions that support Graviton. At this stage, it can be useful to think in terms of software you download (e.g. open source packages, container images, libraries), software you build, and software you procure/license (e.g. monitoring or security agents).

Areas to review:

- Operating system and version (the more recent the better)
- If your workload is container based, check container images you consume for Arm64 support. Keep in mind many container images

now support multiple architectures, which simplifies consumption of those images in a mixed-architecture environment. You can read ECR multiarch support announcement for more details.

- All the libraries, frameworks, and runtimes used by the application and its components.
- The tools used to build, deploy, and test your application (e.g. compilers, test suites, CI/CD pipelines, provisioning tools, and scripts).
- All the tools and/or agents used to deploy and manage the application in production (e.g. monitoring tools or security agents)

There are language-specific sections in the getting started guide with useful pointers to get the best performance from Graviton processors.

The Getting started with AWS Graviton repository will be helpful for this task. It will give you guidance for Operating Systems, Container environments, and various open-source software.

For each component of your software stack, check the version and then check whether they are available for Graviton/arm64. AWS Graviton processors are modern processors and to benefit from their full potential, it is generally recommended to use software versions that are as recent as possible. As an example, Java 8 works perfectly well on Graviton, but we've seen several applications benefiting from upgrading to Java 11 due to arm64-specific optimizations in Java 11 (refer to the Getting started with AWS Graviton repository for details on how to get the best performance on Graviton). It is also worth noting that it is generally simpler to upgrade the dependencies first on x86-64, and then transition to Graviton with the most recent versions of software already in place to reduce the number of variables.

Depending on where you obtain your dependencies, there could be multiple ways to check whether they support Graviton. Some tools, like GCC, call the architecture AArch64, and some others, like the Linux Kernel, call it arm64. When looking for packages in the various repositories, you'll find those different combinations, and sometimes just "ARM."

If you find any software without support for the arm64, please let AWS know by reaching out on the re:Post community support forum.

The main ways to check and places to look include:

- The package repositories of your favorite Linux distributions. The coverage is generally rather comprehensive: Debian, for example has some of the largest package repositories with more than 98% of its packages built for the arm64 architecture, and of the remaining 2%, some are x86 specific or games that are not typically used in a server environment.

- Your container image registry. Amazon ECR now offers public repositories that you can search for arm64 images. DockerHub allows you to search for a specific architecture (Arm64 enabled images).
- On GitHub, you can check for arm64 versions in the release section. However, some projects don't use the release section or only release source archives, so you may need to visit the main project webpage and check the download section. You can also search the GitHub project for "arm64" or "AArch64" to see whether the project has any arm64 code contributions or issues. Even if a project does not currently produce builds for arm64, in many cases, an Arm64 version of those packages will be available through Linux distributions or additional package repositories (e.g. EPEL). You can search for packages using a package search tool, such as pkgs.org.
- In the download section or platform support matrix of your software vendors, look for references to Arm64, AArch64, or Graviton. Software vendor documentation will often list 'platform requirements' which include supported operating system versions and architectures.

Specific to containers you may find an amd64 (x86-64) container image you currently use has become a multi-architecture container image when arm64 support was added meaning there may not be an explicit arm64 container, so make sure you check for both as different projects may choose different ways to vend their container images for both x86-64 and arm64.

Categories of software with potential issues:

- Packages or applications sourced from independent software vendors (ISV's) may not exist for Graviton yet. However, AWS is working with lots of software partners to offer technical guidance to port and optimize their software on Graviton, so the list of available ISV software continues to expand.
- The Python community often produce modules containing low level language code (e.g. C/C++) that needs to be compiled for the Arm64 architecture prior to use on Graviton. While AWS is actively working with the open-source community to ensure the most popular modules are available, in some cases the Python Package Index may lack pre-built binaries for Arm64. To avoid falling back to sub-optimal pure Python versions these modules can automatically be built from source code (See the Python section of the re:Post community support forum for details).

At the end of the first step, you learned enough about Graviton to know how to port your workload and have inventoried your current software stack, so you can start migrating during the second step.

Step 2: Plan your workload transition

TASK 3

Install and configure your application environment

To transition and test your application, you will first need a Graviton environment, so depending on your execution environment, you'll have to:

- Obtain or create an arm64 AMI to boot your Graviton instance(s) from. Depending on how you manage your AMIs, you can either start directly from an existing reference AMI for arm64, or you can build your Golden AMI with your specific dependencies from one of the reference images. See reference list here.
- If you operate a container-based environment, you'll need to build or extend an existing cluster with support for Graviton-based instances. Both Amazon ECS and EKS support adding Graviton- based instances to an existing x86-based cluster. For ECS, you just need to add Graviton-based instances to your ECS cluster, launching them with either the AWS ECS-optimized AMI for arm64 or your own AMI after you've installed the ECS agent. For EKS, you will need to create a node-group with Graviton2-based instances launched with the EKS optimized AMI for arm64.
- Complete the installation of your software stack based on the inventory created in Task 2.

You can support Graviton and x86 instances in the same Auto Scaling Group. This blog details the process using the launch template override.

TASK 4

Build your application(s) and/or container images

Note: if you are not building your application or component parts of your overall application stack, then you may skip this step.

Now that you have an environment available, you can build your application stack.

For applications built using interpreted or JIT'd languages, including Java, PHP, or Node.js, they should run as-is or with only minor modifications.

In many cases your installation scripts can be used as-is or with minor modifications to reference architecture specific versions of components where necessary. The first time through this may be an iterative process as you resolve any remaining dependencies.

The repository contains language-specific sections with recommendations, for example Java, Python, C/C++, Goland, Rust or .Net. Note: if there is no language specific section, it is because there is no specific guidance beyond using a suitably current version of the language as documented here (e.g. PHP Version 7.4+). .NET-core is a great way to benefit from Graviton-based instances. This blog post covers .NET5 performance.

Applications using compiled languages, including C, C++, or Go, need to be compiled for the Arm64 architecture. Most modern builds (e.g. using Make) will work when run natively on Graviton-based instances, however, you'll find language-specific compiler recommendations in this repository: C/C++, Go, and Rust.

Just like an operating system, container images are architecture specific. You will need to build arm64 container image(s). To make the transition easier, we recommend building multi-arch container image(s) that can run automatically on either x86-64 or arm64. Check out the container section of this repository for more details and this blog post provides a detailed overview of multi-architecture container image support, which is considered a best practice for establishing and maintaining a multi-architecture environment.

You will also need to review any functional and unit test suite(s) to ensure you can test the new build artifacts with the same test coverage you have already for x86 artifacts.

At the end of the second step, you have built an environment using Graviton-based instances and installed your application on top of this environment. During the third step, you'll test and ensure you get the expected level of performance.

If you believe you are observing architecture-specific issues, please check the Arm Architecture Reference Manual Armv8 or reach out to us on the re:Post community support forum.

Step 3: Test and optimize your workload

TASK 5

Testing and optimizing your workloads

Now that you have your application stack on Graviton, you should run your test suite to ensure all regular unit and functional tests pass. Resolve any test failures in the application(s) or test suites until you are satisfied everything is working as expected. Most errors should be related to the modifications and updated software versions you have installed during the transition (tip: when upgrading software versions, first test them using an existing x86 environment to minimize the number of variables changed at a time. If issues occur, then resolve them using the current x86 environment before continuing with the new Graviton environment). If you suspect architecture-specific issue(s) please have a look at our C/C++ section, which documents them and gives advice on how to solve them. If there are still details that seem unclear, please reach out to your AWS account team or to the re:Post community support forum.

If after reading the Optimization and Performance Runbook sections and following the recommendations, you don't observe the expected application performance, you can reach out to us on the re:Post community support forum.

TASK 6

Performance testing

With your fully functional application, it's time to establish a performance baseline on Graviton. In most cases, you should expect performance gains. When comparing to existing x86-64 instances, we recommend running tests by fully loading both systems to determine the maximum possible price/performance. You can then determine and configure an appropriate load level for your production environment before performing the deployment.

Important: This repository has sections dedicated to Optimization and a Performance Runbook for you to follow during this stage.

If, after reading the documentation in this repository and following the recommendations, you do not observe expected performance, then please reach out to your AWS account team, or send an email to ec2-arm-dev-feedback@amazon.com with details so we can assist you with your performance observations.

At the end of third step, you will have your application running on top of Graviton-based instances and have established a baseline for the performance. You are now ready to test it in a production environment.

Step 4: Infrastructure and deployment

TASK 7

Update your infrastructure as code

Now that you have a tested and performant application, it's time to update your infrastructure as code to add support for Graviton-based instances. This typically includes updating instance types, AMI IDs, ASG constructs to support multi-architecture (see Amazon EC2 ASG support for multiple Launch Templates), and finally deploying or redeploying your infrastructure.

TASK 8

Perform Canary or Blue-Green deployment

Once your infrastructure is ready to support Graviton-based instances, you can start a Canary or Blue-Green deployment to re-direct a portion of application traffic to the Graviton-based instances. Ideally, you'll run these initial tests in a development environment and load test with traffic patterns as close as possible to production traffic. Monitor the situation carefully to catch any unexpected behavior until your application is running as expected on Graviton, at which point you can determine your transition strategy.

Celebrate

Congratulations! You have completed the Graviton Fast Start program by following the 4-step plan. We understand this project took significant effort and time, and hope you were able to benchmark price performance benefits by using Graviton-based instances for your workloads. We'd love to hear about your experience on the re:Post community support forum.

This is just the beginning of your Graviton adoption journey. If you realized significant price performance gains with your first workload, you can identify more workloads from different AWS services and get even more price performance gains in AWS.

Thanks for using AWS Graviton Fast Start!