AWS GRAVITON2

# Graviton Challenge 4-day adoption plan

## Introduction

AWS Graviton2 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. For a quick overview of Graviton2 processors and EC2 instances powered by Graviton2, visit the AWS Graviton page.

We have seen many customers adopting Graviton2-based EC2 instances with minimal effort. In this document, we have outlined a framework to help you move your workload to Graviton2, using the best practices we have built from working with thousands of customers who have successfully adopted Graviton2-based instances.

This plan is designed so that a single engineer or small team can accomplish all the tasks over a four day period, with each day split into tasks for morning and afternoon. Though it would be good to complete these tasks in four consecutive days, the plan can also be executed across multiple weeks based on your schedule.

If you get stuck at any step, feel free to reach out to Graviton experts for help in the Graviton Challenge Slack channel. By registering for the Slack channel, you also get $25 AWS credits to help you cover the cost of using any Graviton2-based instance for this 4-day plan. You can also use the Graviton2-based T4g.micro instance free for 750 hours per month until 12/31/2021.

### HOW TO IDENTIFY A GOOD TARGET WORKLOAD

A good candidate for Graviton2 migration 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 quickly and easily make any necessary changes as part of this challenge.

If you use third party software, many ISVs already support the Arm64 architecture implemented by Graviton2.

If you use third party software that does not support Arm64, reach out to us on the Graviton Challenge support Slack channel.

Though we have outlined a 4-day plan in this document that is suitable for many applications, it may take you more than four days or less, depending on the complexity of your application. Regardless of the application complexity, the approach and high-level steps described here will remain the same.

## Agenda

The following plan has been organized into approximately 4-hour or half-day time blocks totaling 4-days of elapsed time, and organized into a logical sequence of steps as follows:

- **Day 1 - Learning and exploring**
  - Step 1 - Understand the Graviton2 ecosystem and review key documentation
  - Step 2 - Explore your workload, build its list of dependencies
- **Day 2 - Plan the migration and start porting**
  - Step 3 - Build an environment to migrate your application
  - Step 4 - Bootstrap the application
- **Day 3 - Debugging and optimizing**
  - Step 5 - Testing and debugging
  - Step 6 - Performance testing
- **Day 4 - Infrastructure and deployment**
  - Step 7 - Update your infrastructure as code
  - Step 8 - Canary or Blue–Green deployment

**OPTIONAL PRE-STEP**

Step 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 Step 1, so that you could have their answers by the time you reach Step 2.

## Day 1 - Learning and exploring

**STEP 1**

### Understand the Graviton2 ecosystem and review key documentation (4 hours)

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

Keep on learning by watching re:Invent 2020 - The journey of silicon innovation at AWS to better understand Amazon's long-term commitment to innovate with custom silicon.

Finally, spend some time reading relevant sections of the Graviton Getting Started Guide, which will act as a useful reference throughout this challenge.

**STEP 2**

### Explore your workload, build its list of dependencies (4 hours)

To start the migration, your first task is to build the list of dependencies for your application or workload. Common dependencies include:

- Operating system and version (the more recent the better)
- All the libraries, frameworks, and runtimes used by the application and its components. If your workload is container based, you'll need to locate or build Arm64 versions of those container images (or better still, if arm64 support exists via a multi-architecture container image) — you can read ECR multiarch support announcement for more details.
- The tools used to build, deploy and test it (e.g. compilers, test suites, CI/CD pipelines, provisioning tools, and scripts)
- All the tools and agents used to operate the application in production (e.g. monitoring tools or security agents)

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

For each of the dependencies, check the version and then check whether they are available for Graviton2/arm64. The Graviton2 is a modern and recent processor, and to benefit from its 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 Graviton2, but we've seen several applications benefiting from upgrading to Java 11 due to arm64 specific optimizations in Java 11 (refer to the Graviton Getting Started Guide for details on how to get the best performance on Graviton2). It is also worth noting that it is generally simpler to upgrade the dependencies first on x86-64, and then transition to Graviton2 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 Graviton2. 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."

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, which has some of the largest package repositories, has 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). *Note:* 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.

- On GitHub, you can check whether there's an arm64 version in the release section. Some projects don't use the release section, or only release source code artifacts via GitHub, so you may need to check the main project web page and look in the download section. You can also search the GitHub repository for "arm64" or "aarch64" to see if there are any issues or source code that reference support for the Arm64 architecture. Even if a given project does not produce builds for arm64, it doesn't mean that it isn't ported. In many cases an arm64 version of those packages will be available through Linux distributions.
- In the download section of your software vendor websites, look for arm64- or AArch64-specific packages. Software vendor documentation will often list 'platform requirements' which include supported operating system versions and architectures.
- Packages sourced from independent software vendors (ISV's) may not exist for Graviton2 yet. However, AWS is working with ISV's to offer technical guidance to port and optimize their software on Graviton2 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 Graviton2. 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 Graviton Getting Started Guide for details).

At the end of Day 1, you will have learned enough about Graviton2 to know how to port your workload and have built the list of all your dependencies, so you should be able to start migrating on Day 2.

If you find any software without support for the arm64, please let AWS know by reaching out on the Graviton Challenge support Slack channel.

## Day 2 - Plan the migration and start porting

**STEP 3**

### Build an environment to migrate your application (4 hours)

To port and test your application, you will first need a Graviton2 environment, so depending on your execution environment, you'll have to:

- Obtain or create an arm64 AMI to boot your Graviton2 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. Find a full list here.
- If you operate a container-based environment, you'll need to build or extend an existing cluster with support for Graviton2-based instances. Both Amazon ECS and EKS support adding Graviton2-based instances to an existing x86-based cluster. For ECS, you just need to add Graviton2-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.
- Install all of your application dependencies identified in step 2.

**STEP 4**

### Bootstrap the application (4 hours)

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

For applications built using high-level languages, like Java, Python, or Node.js, the effort should be minimal as they should run as-is or with only minor modifications.

In the Graviton Getting Started Guide you will find sections specific to popular programming languages, listing specific recommendations, for example:

- Java
- Python
- .NET-core is also a wonderful opportunity to run on Graviton2-based instances. Read a blog post on .NET5 performance for those instances.

For applications using lower level languages like C, C++ or Go, they will need to be recompiled.

Again, you will find recommendations in the Graviton Getting Started Guide about specific options depending on the compiler:

- C/C++
- Go

Container images are architecture specific so regardless of the application running in the container you will need to obtain or build a container image that supports the Arm64 architecture. To make the transition easier and to maintain architecture flexibility we recommend building multi-arch container images that can automatically run on both x86-64 and Arm64 architectures.

You will find additional documentation about how to run containers on Graviton2-based instances here and a full explanation of why maintaining multi-architecture support for your container environment is considered a best practice here.

You will also need to review your test suite to check whether it needs to be modified to support the newly introduced architecture as using it to validate your new application environment will keep you busy for most of the next half-day time block.

At the end of Day 2, you should have been able to build an environment using Graviton2-based instances, and have installed your application on top of this environment. On day 3, you'll be able to test it and ensure that you get the expected level of performance.

If you believe you are observing architecture specific issue(s), please check the Arm Architecture Reference Manual Armv8 or reach out to us on the Graviton Challenge support Slack channel.

## Day 3 - Debugging and optimizing

**STEP 5**

### Testing and debugging (4 hours)

Now that you have built your application on Graviton2/arm64, you can run your test suite and ensure that your regular unit and functional tests pass on the new architecture. If there are failures, fix the application or the test suite until they all pass. You can expect most of the errors to be related to the modifications and newer software versions you have installed during the migration (*tip*: to debug errors due to updated software versions, first update on x86 to see if errors occur there, if so resolve them and then transition to arm64/Graviton2 to minimize the number of changes at one time).

If after reading the Graviton Getting Started Guide and following the recommendations you dont observe the expected application performance, you can reach out to us on the Graviton Challenge support Slack channel.

**STEP 6**

### Performance testing (4 hours)

Once you have a fully functional application, it's time to test performance to establish a baseline for your first arm64 version and compare it to x86-64. In most cases, it is expected that you would get performance gains. When comparing to existing x86-64 instances, we recommend to run tests on fully loaded systems. On Graviton2-based EC2 instances, every vCPU is mapped to a physical core, whereas on an x86-64-based EC2 instance, a vCPU is mapped to a hyper-thread. If you run a single copy of a single-threaded test on both architectures, you'll be comparing a full x86-64 physical core (shared by 2 vCPUs) to a Graviton2 physical core (1 vCPU). To run a meaningful test, you should run 1 instance of the test per vCPU. The Graviton Getting Started Guide contains recommendations on optimizations specific to Graviton2.

At the end of Day 3, you will have your application running on top of Graviton2-based instances, and have established a baseline for the performance. You are now ready to test it in a production environment on Day 4.

## Day 4 - Infrastructure and deployment

**STEP 7**

### Update your infrastructure as code (4 hours)

Now that you have a tested and performant application, it's time to update your infrastructure as code to add support for Graviton2-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.

**STEP 8**

### Canary or Blue–Green deployment (4 hours)

Once your infrastructure is ready to support Graviton2-based instances, you can start a Canary or Blue-Green deployment to re-direct a portion of application traffic to the Graviton2-based instances. Ideally, 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 Graviton2, at which point you can determine your transition strategy.

## Celebrate

Congratulations! You have completed the Graviton Challenge by following this 4-day plan. We understand that this project took significant effort and time, and hope that you were able to benchmark significant price performance benefits of using Graviton2-based instances for your workloads.

We'd love to hear about your experience and send you swag for taking the Graviton Challenge. Post about it on social media using the #ITookTheGravitonChallenge hashtag — a video post gets you a Graviton Challenge hoodie and a text post gets you a Graviton Challenge t-shirt.

Additionally, you can enter the Graviton Challenge Contest. All category winners get a pass, flight, and hotel to re:Invent 2021. Best adoption category winners also get a chance to present at re:Invent 2021. All valid entries get $500 AWS credits (limit 500 quantity). To compete, you just need to create a 5-10 minute video about your Graviton Challenge project and complete the submission form with details.

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 to get even more price performance gains in Amazon EC2.

Thanks for taking the Graviton Challenge!