ORACLE

Oracle GraalVM Enterprise Edition

Faster. Smarter. Leaner.

August, 2020 | Version 1.01

PURPOSE STATEMENT

This document provides an overview of features and enhancements included in release 20.1. It is intended solely to help you assess the business benefits of upgrading to 20.1 and to plan your I.T. projects.

DISCLAIMER

TABLE OF CONTENTS

MODERN APPLICATION CHALLENGES

As Marc Andreessen famously observed, "Software is eating the world"[1]. Whether for development of external customer-facing services, or for automating internal business processes, the use of software continues to accelerate. In an endeavor to scale applications with ever increasing demands, and keeping up with business SLA guarantees, enterprises are incurring higher information technology bills. The situation is exacerbated on public cloud platforms where the charges are based on both CPU and memory usage. Hence there is a persistent need to find innovative ways to get more out of existing investments.

This quest for higher efficiency has also led to a shift in the way the companies build software systems. Whether managing their own data centers (on-premises) or running their workloads on public cloud, they are migrating away from the N-tier enterprise platforms that have dominated software architecture for the last two decades. They are progressively moving towards horizontally-scalable, container-based platforms and microservices—key elements of a cloud-native architecture.

Regardless of the deployment choice, to curtail IT expenses, applications must run faster and consume fewer resources. Oracle GraalVM Enterprise Edition is an application runtime that addresses these challenges. Built on trusted and secure Oracle Java SE, GraalVM Enterprise accelerates application performance while consuming fewer resources—improving application efficiency and reducing compute costs.

Incorporating over a decade of research and development innovations into advanced optimizing compiler technology, GraalVM Enterprise is the ideal platform for modern application deployments.

Key Features

- Leverages new optimization algorithms to improve performance of enterprise applications
- A high-performance runtime for modern microservices
- Compiles Java applications ahead of time into native images to improve startup and memory footprint
- Extends applications with libraries from other supported languages without performance penalties
- Runs native languages like C/C++ in a managed mode on the JVM

GRAALVM ENTERPRISE

GraalVM Enterprise offers features for accelerating the performance of existing Java applications and for building microservices. It includes advanced optimizing compiler technology to provide a high performance Just-In-Time (JIT) compiler that can be used to accelerate the performance of any Java and JVM-based application—without any code changes!

GraalVM Enterprise also incorporates its optimizing compiler technology into an advanced Ahead-Of-Time (AOT) Native Image compiler that translates Java and JVM-based applications into native platform executables. These native platform executables are smaller, start nearly instantaneously, and consume a fraction of the resources of the same Java application running on a JVM, making them ideal for cloud deployments and microservices.

Built on Open Source

Oracle GraalVM Enterprise Edition is built on the Oracle-led GraalVM open source project. The open source GraalVM community includes participants from many of the leading companies and has an advisory board with members from industry and academia including Shopify, VMWare, Red Hat, Microsoft, Twitter, Amazon, and more.

The purpose of the open source GraalVM project is to build and share a platform for language runtime innovation. Its success can be seen by the stature of the companies and universities who are participating, and by the number of projects that are using GraalVM. GraalVM Enterprise includes all the features of the open source Community Edition, along with additional performance and resource optimizations, as well as full enterprise grade support from Oracle. For example, GraalVM Enterprise's compiler includes 27 additional patented optimizations that result in faster code and lower CPU and memory requirements.

Companies looking for the fastest, smartest, and leanest runtime for their applications choose GraalVM Enterprise.

Built on Oracle Java SE

Celebrating its 25th anniversary in 2020, Java continues to be the leading application development language. Enterprises rely on Oracle Java SE for reliability, security, and regular updates. While GraalVM Enterprise provides significant benefits for Java applications, it is built on proven Oracle Java SE.

GraalVM Enterprise releases include all Oracle Java critical patch updates (CPUs) which are released on a regular schedule to remedy defects and known vulnerabilities. GraalVM Enterprise customers get 24/7 access to the experienced Oracle GraalVM Enterprise Edition support team who work closely with the Oracle Java support team so you can:

- Log and resolve GraalVM Enterprise issues quickly and efficiently
- Reduce time to resolution and minimize support costs
- Maximize application uptime

Java Facts

- 98% of Fortune 100 run Java
- 45 billion JVMs globally
- #1 programming language
- #1 developer choice in the cloud

ACCELERATING APPLICATION PERFORMANCE

Without any code changes, GraalVM Enterprise can improve the performance of any Java application and any application that runs on the Java Virtual Machine. Faster application execution provides two benefits:

First, it reduces the response time for user requests, whether interactive or via RES. Applications running on GraalVM Enterprise exhibit lower latency, which is crucial when you remember that forty percent of consumers abandon web pages and shopping carts if the response time is over three seconds.[2]

Second, applications that run faster free up CPU and memory sooner, allowing them to handle other requests or other applications running on the same server. In data centers with ever-increasing workloads, being able to service more requests with the same computing infrastructure reduces the need to purchase additional hardware. Thus, GraalVM Enterprise's reduction of required compute resources can lower capital cost expenditures on premise and lower operation costs on cloud.

Faster Code—Advanced Compiler Optimizations

- 27 patented optimizations
- Aggressive inlining—increases optimization opportunities
- Polymorphic inlining—faster virtual method dispatch
- Partial Escape Analysis—eliminate or delay object allocations
- Lower memory and CPU usage—reduced garbage collection
- Less garbage means less time collecting thanks to advanced compiler optimizations
- Less GC means more CPU for application code—higher throughput

The GraalVM Enterprise Compiler

The GraalVM Enterprise compiler includes 62 separate compiler optimization algorithms (called "Phases"), of which 27 are patented, including new techniques for vectorizing complex programs, large-scale escape analysis, and code specialization. In general, JVM languages (e.g., Java, Scala, and Kotlin) will see around a 20%–30% speedup using GraalVM Enterprise when compared with Java SE 8. Many large Scala applications can show 30% to 40% performance improvements. Code using more modern Java styles (like Streams and Lambdas introduced in JDK 8) and more abstractions will see even greater speedups. Low level C-like code or code that is bottlenecked on things like I/O, memory allocation, or garbage collection, will see less improvement. Benchmarks are not necessarily a perfect indicator of the performance of a typical application, but to date the most accurate performance predictions for the JVM come from the "Renaissance"[3] benchmark, which has been developed by Prague's Charles University in conjunction with Oracle Labs.

Figure 1 shows the results for the Renaissance benchmarks for GraalVM Enterprise which shows an average increase of 55% over JDK 8 with some benchmarks running over 4x faster.

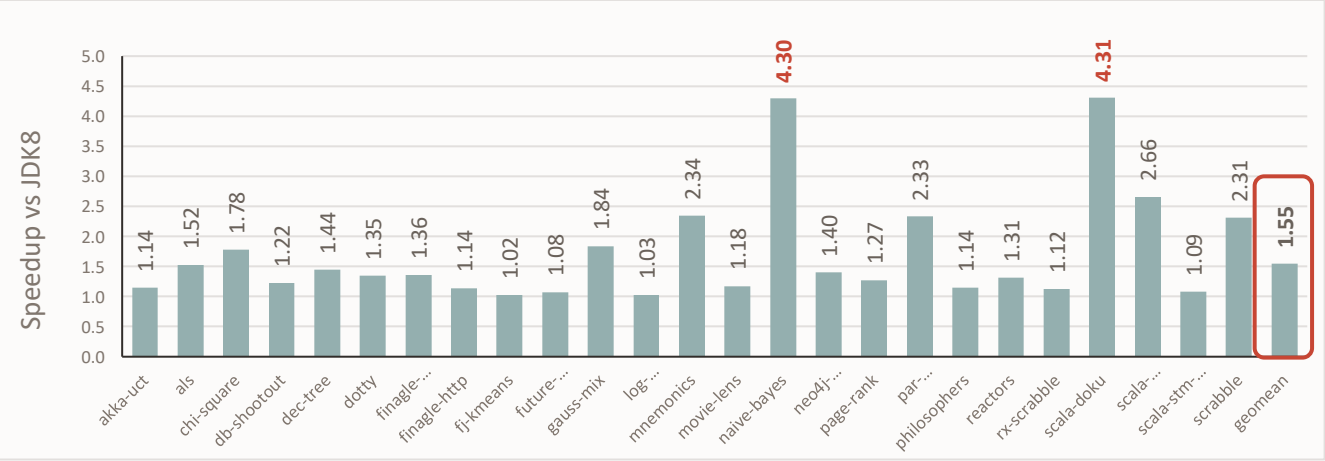

Figure 1: GraalVM Enterprise Renaissance Benchmark Performance

The paper motivating the Renaissance benchmark was recently accepted by PLDI[4] (Programming Language Design and Implementation), the world's most prestigious programming language conference. One of the key reasons the paper is considered new research is that the authors were able to show that Renaissance is more representative of larger, real-world applications, and not just built from a majority of small, microbenchmarks that are easy to hand-optimize in ways that render the benchmark results unrepresentative for larger applications. To illustrate, **Figure 2** and **Figure 3** show that Renaissance benchmarks have more hot code and many more hot methods than classical JVM benchmarks like SPECjvm2008.[5] This shows that the Renaissance benchmarks are more reflective of real-world applications where complex tasks are performed repetitively. In addition, older benchmarks like SPECjvm and DaCapo don't include features from more modern versions of Java such as Streams and Lambdas, which were introduced in JDK 8.

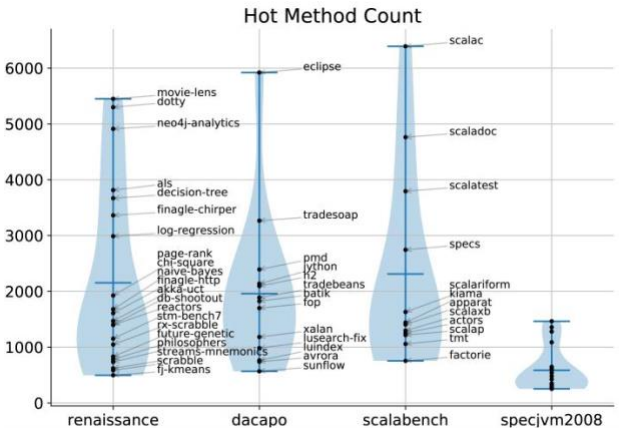

Figure 2: GraalVM Enterprise Renaissance Benchmark Hot Method Count.

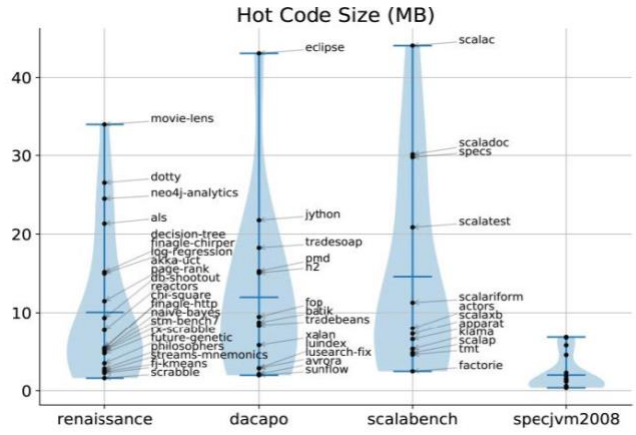

Figure 3: GraalVM Enterprise Renaissance Benchmark Hot Code Size

The GraalVM Enterprise compiler shines even brighter for the Scala language, which typically has more abstractions for the compiler to work with, allowing for more speedup. The Scalabench suite is a good predictor of Scala performance across a wide range of Scala applications. Twitter, the most well-known Scala user, has been using the GraalVM Community compiler for many years, gaining significant speedups that have saved the company millions of dollars a year in annual infrastructure expenditures. The Scalabench run for GraalVM 20.1 is shown in **Figure 4**. As you can see, GraalVM Enterprise 20.1 significantly outperforms the base JDK8 in the vast majority of tests, averaging about 34% faster.

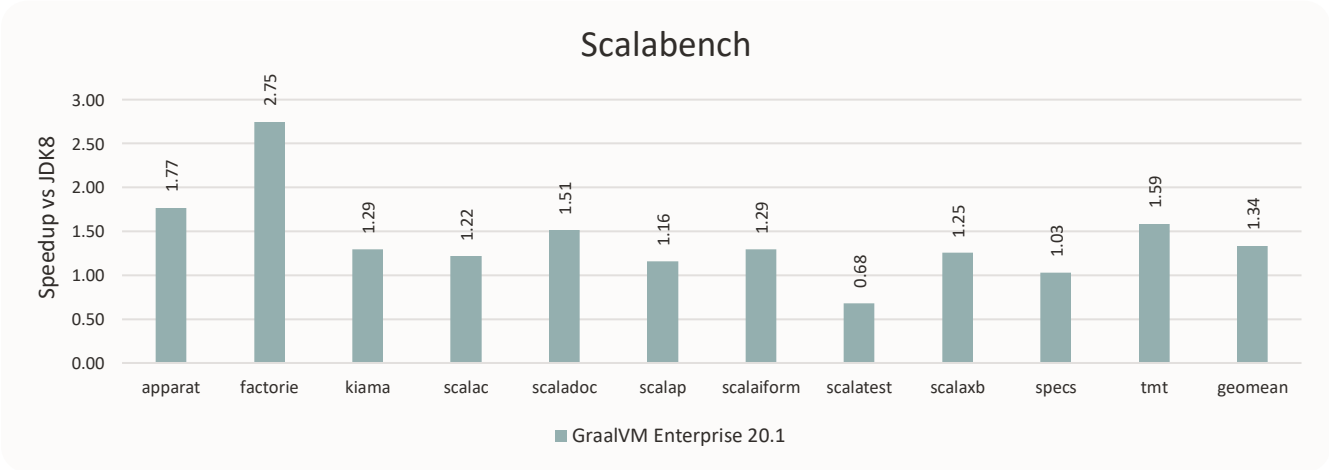

Figure 4: GraalVM Enterprise and Community Scalabench Performance relative to JDK8

What Makes GraalVM Enterprise Faster

Benefitting from over a decade of compiler research and development by Oracle Labs, as well as the experiences of the industry-leading GraalVM community, GraalVM Enterprise runs applications faster because it generates highly optimized machine code. Compiler phases including aggressive inlining, polymorphic inlining, and other techniques, make code more efficient. Thanks to advanced memory-allocation optimizations, such as partial escape analysis and scalar replacement[6], GraalVM Enterprise is capable of optimizing away many object allocations. Consequently, an application running on GraalVM Enterprise needs to spend less time doing automatic memory management and garbage collection. From a production deployment perspective, this means that an application running on GraalVM Enterprise can achieve better performance with less memory, which is particularly important for cloud environments.

Apache Spark on GraalVM Enterprise

As companies that are deploying Apache Spark know well, when processing large amounts of data, performance is absolutely critical. While there are many sophisticated performance tuning options and techniques available in Apache Spark, one of the easiest ways to improve overall performance is to run it on GraalVM Enterprise. The Renaissance benchmark suite's Apache Spark benchmarks show an average workload execution time reduction of 1.6x, with one benchmark running 4x faster!

Peak performance is the performance of an application after the Java Virtual Machine warms up, and the Just-In-Time compiler-generated machine has been fully optimized. The following graphs compare the peak performance of OpenJDK 8 and GraalVM Enterprise. We'll take OpenJDK 8 as the baseline, with the y-axis showing the normalized performance, so higher is better. Overall, GraalVM Enterprise achieves higher peak performance than OpenJDK 8 on all Apache Spark benchmarks. The performance of Apache Spark on GraalVM Enterprise improves by an astonishing mean of 60% (1.6x normalized). In **Figure 5**, the performance of GraalVM Enterprise can be seen to exceed OpenJDK 8 on the naïve-bayes benchmark by about 4x.

GraalVM Enterprise allows your Apache Spark workloads to run faster and consume fewer resources without any additional changes to your database configuration or application code.

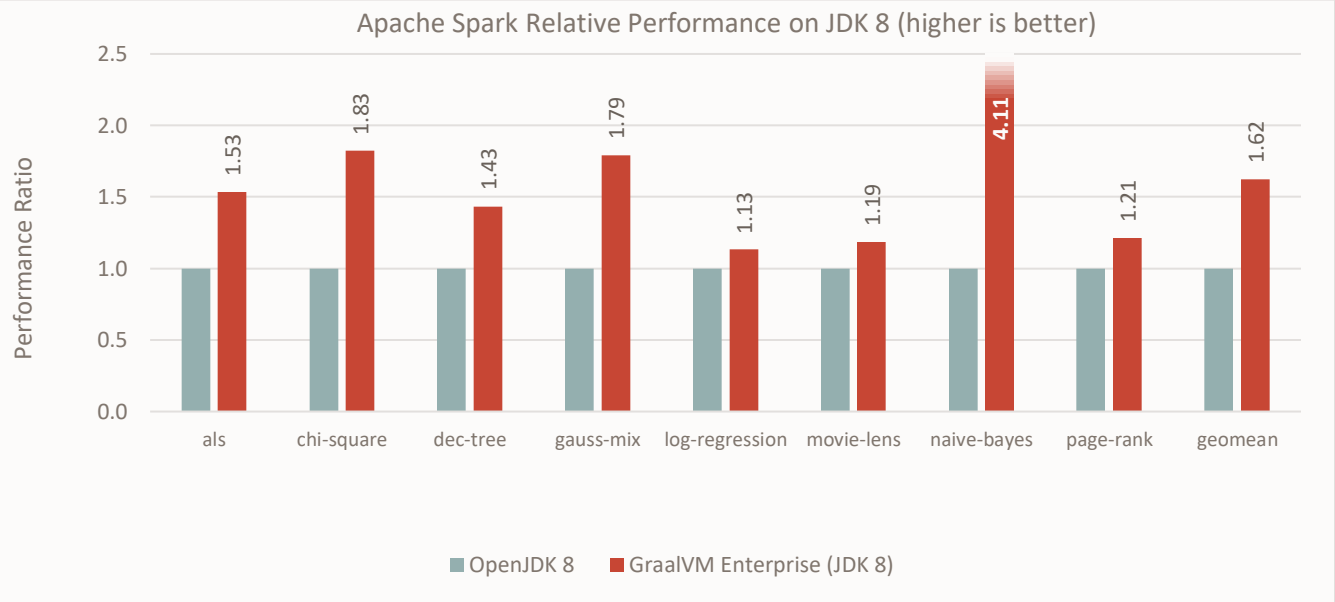

Figure 5: *Apache Spark Performance of GraalVM Enterprise Relative to OpenJDK8*

Accelerating WebLogic Applications

With the certification of WebLogic Server versions 12.2.1.4 and 14.1.1 on GraalVM Enterprise, Java Enterprise Edition users can take advantage of the performance benefits GraalVM Enterprise provides. In **Figure 6,** performance is up to 30% faster on some benchmarks with typical applications seeing up to 10% improvement. Each column in Figure 6, below, represents a benchmark in the suite and shows the range of performance of each benchmark from the minimum performance improvement to the highest performance improvement.

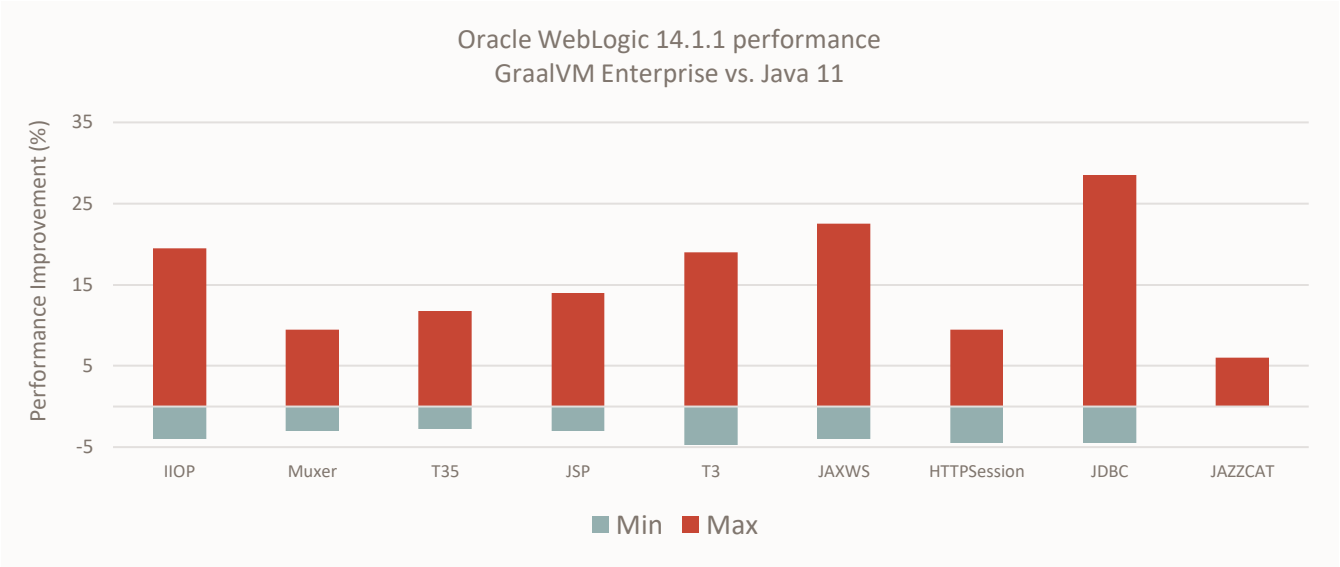

Figure 6: *WebLogic Server Performance on GraalVM Enterprise*

Running WebLogic Server on GraalVM Enterprise is as simple as changing its Java Home, as seen in **Figure 7**. No changes to application code or configuration are necessary. Once running, GraalVM Enterprise's optimizing compiler will produce faster Just-In-Time compiled machine code and eliminate unnecessary object allocations, thus reducing garbage collection costs.

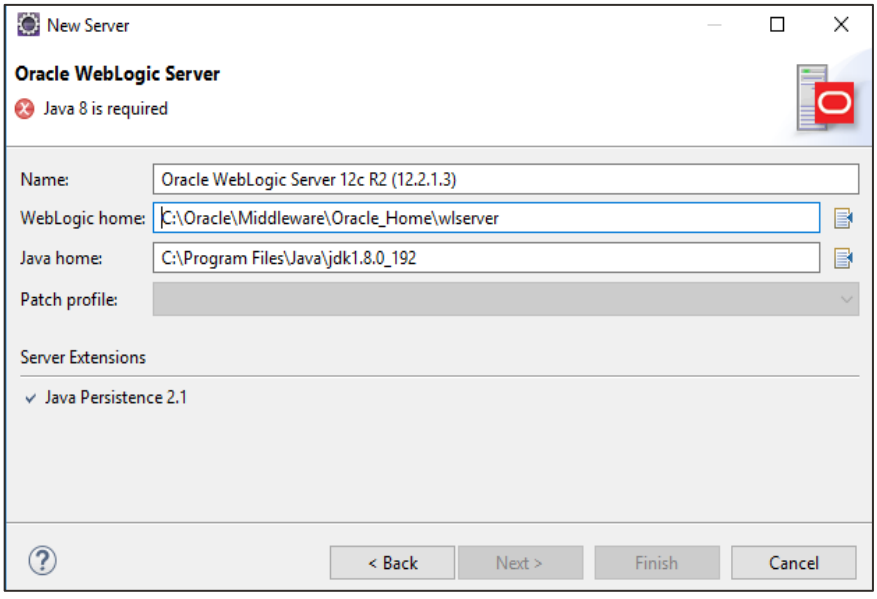

Figure 7: *Running Oracle WebLogic Server on GraalVM Enterprise*

THE IDEAL MICROSERVICES RUNTIME

Microservices are increasingly being used to build modern applications. This shift from large applications with multiple services to a collection of smaller single service applications imposes new requirements on applications. Microservices applications are highly distributed, which allows for significant redundancy and reliability. They are designed to execute effectively in today's modern cloud environments. To handle peak demand, it's also necessary to able to scale up and down rapidly. However, many microservice deployments are unable to scale up or down quickly and it can take seconds or even minutes to add additional capacity to meet demand spikes.

GraalVM Enterprise's Native Image utility can compile Java applications into native binary executables that have incredibly fast start times since program initialization can be done at build time and the application is already compiled. They will likewise have much smaller footprints since the JVM and JIT compiler don't have to be distributed.

These qualities have led to GraalVM Native Image being the preferred way to run applications built by all the major microservice frameworks including Helidon, Micronaut, Quarkus, and Spring Boot.

On a popular framework benchmark, GraalVM Enterprise Native Image provides instant and consistent throughput, as seen in **Figure 8**. This results in an initial transactions-per-second rate that is 1500% higher than a JIT compiled application, allowing microservices to perform at their peak, immediately. This eliminates the need to keep services running for peak-time only demands.

If the application is run for some time and the JIT compiler has time to warm up, it will achieve higher peak performance, e.g., 16% in the benchmark. So, while Native Image is ideal for fast starting and relatively short-lived microservices, JIT is a better option for traditional long-running Java workloads.

GraalVM Enterprise Native Image Advantages

- Minimized Attack Surface as the Native Image compiler removes unused classes, methods, and fields.
- Profile Guided Optimization allows you to profile your application to achieve maximum performance even when compiled Ahead-Of-Time.
- Compressed Pointers reduce application memory footprint by using 32-bit references to Java objects on 64-bit architectures.
- G1-based Garbage Collector supports large heaps and pause time goals while achieving high throughput.
- Isolates support separate heaps for independent processes to improve isolation and reduce garbage collection.

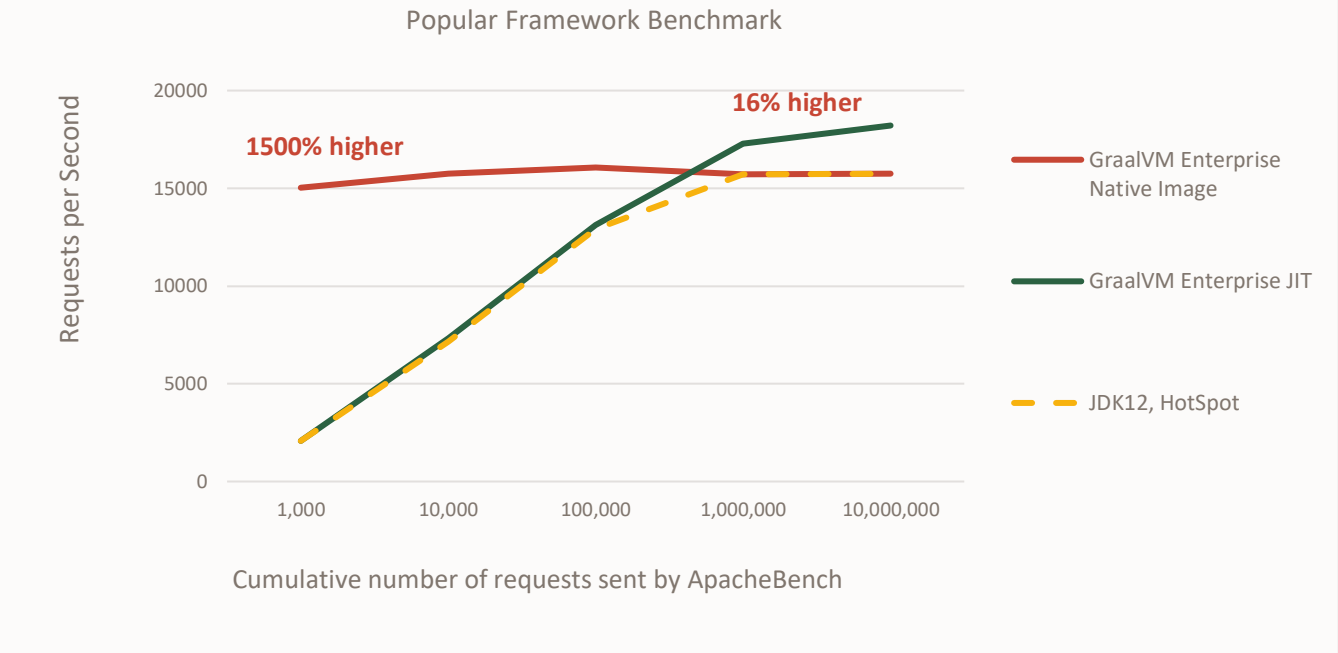

Figure 8: *Throughput over time of GraalVM Enterprise using JIT compilation and Native Image (AOT) compilation vs JDK12.*

Microservice Startup, CPU, and Memory Usage with GraalVM Enterprise

Let's take a look at a simple microservice to see the impact of native image compilation on application resource consumption. We'll use the Micronaut Framework for our application that calculates prime numbers. We'll start up the service and then ask it to give us back all prime numbers between 2 and 100. We'll do this three times.

To ensure the service is ready, we query it approximately every 0.02 seconds until we get a valid response. We then loop three times sleeping for one second before making each request for primes.

We do this first using Just-In-Time compilation with JDK8, the results of which are the chart on the left in **Figure 9**. Then, we repeat the process, but this time we use the GraalVM Enterprise Native Image (JDK8) compiled version of the same Micronaut application. The results of this run are depicted in the chart on the right in Figure 9.

The JIT version of the application takes approximately two seconds (2044ms) to start and return a valid response. That first request is the one we use to confirm the application is ready. Then there is one second or more to compile additional code being executed on the server side in preparation to accept the second request. During those three seconds, the process takes up as much as 15,000% of the CPU in spikes and averages about 3000%. It also climbs in memory usage until it plateaus at about 200MB.

Conversely, the Native Image compiled version of the application takes less than 1 second to startup (744ms) and return the first request that we use to confirm the process is ready. During this time, the service consumes approximately 50% CPU and plateaus at about 40MB of memory.

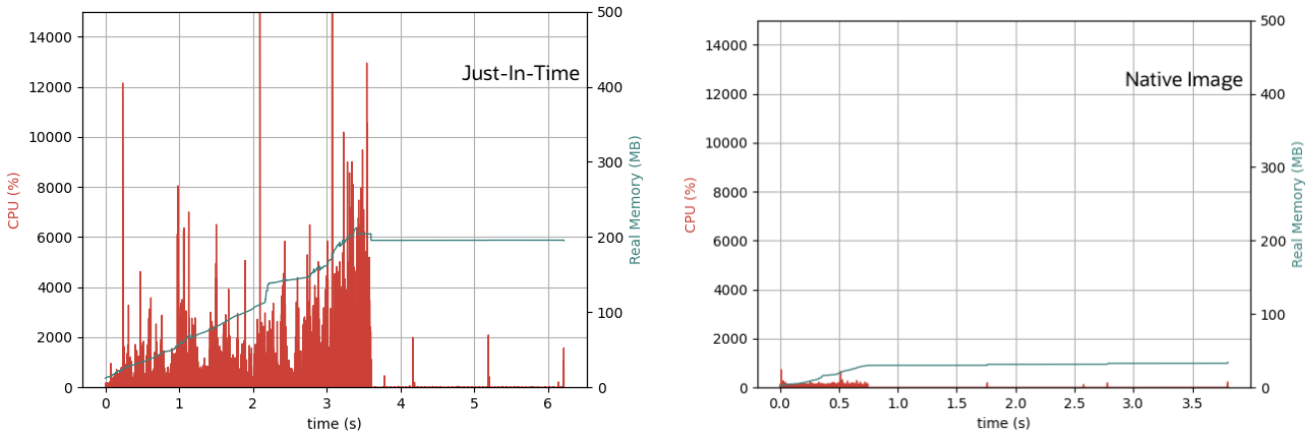

Figure 9: JIT vs. Native Image (AOT) starting up and serving two requests in the first ten seconds

IMPROVED PRODUCTIVITY WITH MULTI-LANGUAGE SUPPORT

Java is the most popular language for building enterprise applications, but JavaScript, Python, Ruby, and other programming languages are also frequently used. Many of these languages have been either designed for or have carved out a niche in specific areas, e.g., machine learning, and have libraries that provide unique and valuable features. Hence, developers often use different languages for specialized applications.

Ideally, a developer should be able to incorporate libraries from any language into their Java application, allowing them to take advantage of the unique capabilities of that language or library. Or, if their primary language is not Java, developers should be able to incorporate Java libraries into their non-Java applications. Traditionally this has been difficult to do. It has typically introduced significant performance penalties that have made it impractical—until now.

GraalVM Enterprise provides high performance runtime support for a number of languages beyond Java, along with the ability to have different languages and libraries interoperate with no performance penalty. This support improves developer productivity by letting them use the right language or library for a given task.

Key to GraalVM Enterprise's polyglot support is language compliance. For each of the supported languages, GraalVM Enterprise strictly adheres to the specification for each language and runs their compliance test suites to ensure compatibility. For example, as seen in **Figure 10**, GraalVM Enterprise's JavaScript support is ECMAScript 2019 compliant and continues to implement the new features being added to the language in the ECMAScript 2020 specification.[7]

					Servers/runtimes							
65%	94%	59%	97%	98%	98%	98%	98%	98%	25%	95%	7%	98%
Echo JS	XS6	JXA	Node >=8.10 <9[3]	Node >=10.9 <11[3]	Node >=12.11 <13[3]	Node 13.0-13.1[3]	Node >=13.2 <14[3]	Node 14.0+[3]	DUK 2.3	JrS 2.3.0	JJS 1.8	GraalVM 20.1.0[4]
0/2	2/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	2/2	0/2	0/2	0/2
4/7	7/7	0/7	7/7	7/7	7/7	7/7	7/7	7/7	0/7	7/7	0/7	7/7
3/5	5/5	0/5	5/5	5/5	5/5	5/5	5/5	5/5	0/5	5/5	0/5	5/5
10/15	15/15	11/15	15/15	15/15	15/15	15/15	15/15	15/15	0/15	15/15	0/15	15/15
5/6	6/6	5/6	6/6	6/6	6/6	6/6	6/6	6/6	4/6	6/6	0/6	6/6
7/9	9/9	8/9	9/9	9/9	9/9	9/9	9/9	9/9	0/9	9/9	0/9	9/9
2/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	0/4	4/4
6/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	0/7	7/7	0/7	7/7
2/6	2/6	0/6	6/6	6/6	6/6	6/6	6/6	6/6	0/6	6/6	0/6	6/6
12/22	21/22	19/22	22/22	22/22	22/22	22/22	22/22	22/22	0/22	22/22	0/22	22/22
14/24	24/24	21/24	24/24	24/24	24/24	24/24	24/24	24/24	0/24	24/24	0/24	24/24
12/25	23/25	18/25	25/25	25/25	25/25	25/25	25/25	25/25	0/25	25/25	0/25	25/25
3/4	3/4	3/4	4/4	4/4	4/4	4/4	4/4	4/4	3/4	1/4	0/4	4/4
2/2	2/2	0/2	2/2	2/2	2/2	2/2	2/2	2/2	1/2	2/2	0/2	2/2
10/18	18/18	12/18	18/18	18/18	18/18	18/18	18/18	18/18	4/18	18/18	12/18	18/18
10/16	14/16	0/16	16/16	16/16	16/16	16/16	16/16	16/16	0/16	16/16	10/16	16/16
Yes	Yes	?	Yes	?	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes

Figure 10: GraalVM ECMAScript Compliance

GraalVM Enterprise's Ruby support is unique in being the only runtime other than the standard MRI (CRuby) to pass all 2180 of the RubyGems test with no exclusion. Compatibility along with excellent performance is the reason companies with huge Ruby estates like Shopify are investing in GraalVM Ruby.[8]

GraalVM continues to add support for new languages and technologies including Python, LLVM bit code, and WebAssembly.

GRAALVM ENTERPRISE SECURITY

When GraalVM Enterprise's Native Image Utility Ahead-of-Time compiles Java code into a native executable, it only includes the code required to execute the application, greatly reducing the application attack surface. It does this by statically analysing the application code under a "closed world" assumption, which means the application code, its dependencies, dependent JDK libraries, and VM components, are analyzed to determine which classes and methods are actually reachable and used during execution. The output of this analysis is that

only the “reachable bytecode" is passed as input to the GraalVM compiler. This process can eliminate as much as 96% of the Java code from the JDK and other libraries in the resultant native executable.

Since there’s less code, there’s less for an attacker to exploit. Additionally, with Ahead-Of-Time compiled native executables, there’s no dynamic class loading that can be exploited in a de-serialization attack. A native executable won’t be impacted by a vulnerability in a class that might have been part of a Java library that the application included but never actually used. This means the application will not need to be patched as frequently to address vulnerabilities in third party dependencies.

GRAALVM ENTERPRISE: AT THE CENTER OF INNOVATION

GraalVM started with an idea to build a Java complier in Java that would be easy to maintain and offer improved performance. As GraalVM advanced it took on visionary ideas and capabilities such as Ahead-Of-Time compilation for Java and multilingual support. But the core goal has always been, and will continue to be, accelerating the performance of mission critical applications. It is this relentless focus on performance that has made every release of GraalVM faster than the last.

GraalVM also continues to provide excellent language specification compliance for all supported languages, and tracks changes through Oracle’s participation in a number of standards bodies like Ecma International.[9] GraalVM innovations are in turn influencing specifications, like the recent announcement of the OpenJDK’s Project Leyden, which seeks to provide a standard specification for Java binary executables pioneered by the Native Image feature.

GraalVM Enterprise is a foundation for innovation for both large enterprise applications and microservices. The unique set of capabilities that GraalVM Enterprise provides has led to it being integrated into Oracle NetSuite for user scripting, and into Oracle Coherence to bring advanced data grid and in-memory caching technology to JavaScript and Node.js, and other languages. It’s also being integrated into the Oracle Database to provide multilingual support for stored procedures to improve its usability for modern full stack developers. GraalVM Native Image has also become the de facto standard platform for microservices with support from all leading frameworks including Micronaut, Helidon, Spring Boot, Quarkus, and Tomcat.

GETTING STARTED

GraalVM Enterprise is easy to try out. Built on Oracle Java SE, GraalVM Enterprise is a drop-in replacement for your current JDK. Just download from the Oracle Technology Network, unzip, set your JAVA_HOME variable, and go. GraalVM Enterprise is free for development and evaluation usage and can be downloaded from the Oracle Technology Network. GraalVM Enterprise is free for production applications (including support) in Oracle Cloud and is built into the Oracle Cloud Developer Image, which includes the latest tools, OCI SDKs, database connectors, and more.

The full documentation is available on the Oracle Help Center

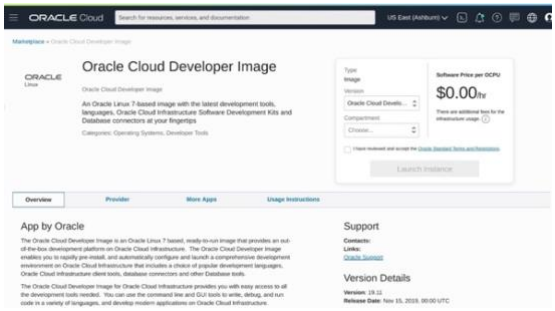

Figure 11: *Oracle Cloud Developer Image with GraalVM Enterprise*

REFERENCE

1 Andreessen, Marc. “Why Software is Eating the World.” a16z.com. https://a16z.com/2011/08/20/why-software-is-eating-the-world

2 Patel, Neil. “How Loading Time Affects Your Bottom Line.” neilpatel.com. https://neilpatel.com/blog/loading-time

3 Renaissance Suite. “A Modern Benchmark Suite for the JVM.” https://renaissance.dev

4 SIGPLAN. “Program Language Design and Implementation Conference, 2019.” https://pldi19.sigplan.org/home

5 Prokopec, Aleksandar, Andrea Rosà, David Leopoldseder, Gilles Duboscq, Petr Tůma, Martin Studener, Lubomír Bulej, Yudi Zheng, Alex Villazón, Doug Simon, Thomas Wuerthinger, Walter Binder. “On Evaluating the Renaissance Benchmarking Suite: Variety, Performance, and Complexity.” arXiv.org. https://arxiv.org/pdf/1903.10267.pdf

6 Stadler, Lukas, and Thomas Würthinger, Hanspeter Mössenböck. “Partial Escape Analysis and Scalar Replacement for Java.” CGO '14: Proceedings of Annual IEEE/ACM International Symposium on Code Generation and Optimization (February 2014):165-174. http://www.ssw.uni-linz.ac.at/Research/Papers/Stadler14/Stadler2014-CGO-PEA.pdf

7 “ECMAScript Compatibility Table.” https://kangax.github.io/compat-table/es2016plus

8 Chen, Carol. “Optimizing Ruby Lazy Initialization in TruffleRuby with Deoptimization.” Shopify. https://engineering.shopify.com/blogs/engineering/optimizing-ruby-lazy-initialization-in-truffleruby-with-deoptimization

9 Yurenko, Alina. “Oracle Becomes an Ecma TC39 Member.” Oracle. https://blogs.oracle.com/graalvm/oracle-becomes-an-ecma-tc39-member

CONNECT WITH US

Call +1.800.ORACLE1 or visit oracle.com.
Outside North America, find your local office at oracle.com/contact.

blogs.oracle.com facebook.com/oracle twitter.com/oracle

Oracle GraalVM Enterprise Edition
August, 2020