JVM

De smaakmaker van Java

Stel, je bent op zoek naar een nieuwe auto. Je gaat afwegen welke auto het beste bij je past. Je maakt een keuze op basis van parameters zoals de infrastructuur waar je op gaat rijden, maak je vaak lange of korte ritten, en hoeveel zitplaatsen of ruimte heb je nodig? Wist je dat je dit concept ook kunt toepassen op het kiezen van een distributie van Java? In dit artikel gaan we daar uitgebreid op in. Het kiezen van de juiste distributie met bijbehorende JVM kan namelijk een behoorlijke impact hebben op hoe jouw Java-applicatie zich gedraagt.

Menig Java developer zal niet anders gewend zijn dan met Java aan de slag te gaan door de Java Development Kit (JDK) te downloaden van de Oracle (referentie 1) website. Hier is niks mis mee, omdat je zo direct volledig aan de slag kan.

OpenJDK is de open source codebase voor vele andere JDK distributies, waaronder de zojuist genoemde van Oracle zelf. De OpenJDK is op zichzelf geen distributie, maar source-code. Voor providers van binaire distributes, gebaseerd op de OpenJDK sources, is het mogelijk om hun compatibiliteit te testen met de Java Compatibility Kit (JCK of TCK voor Java SE). Dit maakt het mogelijk voor vendors om compatible (drop-in replacements) te zijn met Hotspot gebaseerde distributies van Oracle. Deze zogenaamde downstream distributies van OpenJDK kunnen snel voor hun klanten hun eigen patches en optimalisaties doorvoeren. Hier gaan we later in het artikel dieper op in met voorbeelden. Vaak geven deze distributies hun patches ook weer terug (upstream) aan de OpenJDK.

Hieronder wordt een representatieve, maar niet complete, lijst van downstream distributies van 'general purpose' JDK's en hun JVM's beschreven.

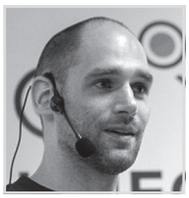
Oracle "HotSpot"

Dit is de reference implementatie van Oracle die sinds het ontstaan van Java doorontwikkeld is in de OpenJDK code base (referentie 3). Mark Reinhold, de Chief Architect van Java bij Oracle, heeft in een presentatie meegedeeld dat er naar schatting al meer dan een millennium aan werk in manuren in deze JVM zit. Dit geeft een idee van de complexiteit van de interne werking van een JVM. Standaard wordt deze JVM geleverd in de Oracle JDK (referentie 1) en Oracle OpenJDK distributie (referentie 2). De Oracle JDK versie is hun commerciële versie waar ze support op leveren en de OpenJDK distributie is geleverd als open source variant.

Eclipse "OpenJ9"

Deze JVM is gebaseerd op de IBM J9 JVM, en wordt sinds 2017 nog steeds door onder andere IBM ontwikkeld als open-source project onder de Eclipse Foundation. Vandaar dat de JVM in die omgeving de naam Eclipse OpenJ9 draagt (referentie 5). Ondanks dat deze JVM volledig compliant is aan de JVM specificaties is deze niet gebaseerd op de OpenJDK codebase. Hij is volledig gebouwd door IBM in de tijd dat Java nog niet open source was. Vanwege het feit dat IBM deze JDK in veel van hun eigen middleware producten gebruikt, heeft deze JVM een reputatie kunnen opbouwen op het gebied van hoge prestaties, schaalbaarheid en betrouwbaarheid met als paradepaardje het geoptimaliseerde geheugengebruik.

Ivo Woltring is Software Architect en Codesmith bij Ordina JTech en houdt zich graag bezig met nieuwe ontwikkelingen in de software wereld.



Edwin Derks is Software Architect en CodeSmith bij Ordina JTech en heeft een passie voor alles wat met Java en cloud-driven development te maken heeft. Hij is ook contributor voor de Jakarta EE en MicroProfile communities, naast het spreken op conferenties en schrijven van artikelen.

AdoptOpenJDK

AdoptOpenJDK levert downstream binaries van de OpenJDK, voor verschillende besturingssystemen (referentie 4). AdoptOpenJDK binaries zijn dus grotendeels gelijk aan de Oracle OpenJDK distributie. Als er security fixes zijn die geïmplementeerd worden in de OpenJDK sources zal AdoptOpenJDK daar binaries voor bouwen. Hetzelfde doen ze ook voor Eclipse OpenJ9 binaries. AdoptOpenJDK als bedrijf levert verder geen support op de verschillende versies.

Azul Systems "Zulu"

Azul onderhoudt binaire downstream versies van de OpenJDK (zulu) en biedt hier support op, ook als Oracle al gestopt is met support leveren, bijvoorbeeld voor Java SE 1.6. Patches zullen dus worden gemaakt op basis van Service Level Agreements (SLA) wat meteen ook hun verdienmodel is. Ze blijven ook nieuwe ontwikkelingen toepassen (back-porten) op oudere versies. Zo is dit momenteel de enige Java SE 1.8 met een Java Flight Recorder (JFR) en TLS 1.3 support.

Amazon Corretto

Corretto is een door Amazon onderhouden downstream distributie van de OpenJDK. Alle AWS instanties die Java draaien gebruiken standaard Corretto. OpenJDK is de source van deze distributie en Corretto voert zijn eigen patches hier op uit. Amazon, met al zijn klanten, is een groot gebruiker van Java en Corretto is hierdoor een stabiele distributie geworden. Ze zijn een nieuwe distributeur (sinds eind 2018) en bieden multi platform downloads aan van Corretto 8 en 11. Corretto is uniek, omdat ze kosteloos zogenaamde Long Term Support (LTS) leveren op deze versies (referenties 8 en 9). De LTS voor Corretto 8 loopt in ieder geval tot 2023 en is een drop-in replacement voor de op HotSpot gebaseerde JDK's.

Oracle "GraalVM"

GraalVM is een universele virtuele machine voor het uitvoeren van applicaties die zijn geschreven in JavaScript, Python, Ruby, R, JVM-gebaseerde talen zoals Java, Scala, Groovy, Kotlin, Clojure en LLVM-gebaseerde talen zoals C en C++. GraalVM verwijdert de isolatie tussen programmeertalen en maakt interoperabiliteit in een gedeelde runtime mogelijk. Daarnaast bevat GraalVM een hele lijst van features die je niet terugvindt in een andere JVM. Een van de bekendste features is de ahead-of-time compilation (AOT) gebaseerde JIT waarmee deze JVM beter kan presteren. Deze feature gebruik je als je GraalVM gebruikt als een conventionele JVM.

Daarnaast heb je met GraalVM namelijk ook de mogelijkheid om je app helemaal native te compileren door middel van AOT compilatie (referentie 11). Hiermee wordt machine geoptimaliseerde code gegenereerd van je app, zoals een standaard native compiler die zou produceren. Deze kun je dan draaien als een native executable zonder JVM. Hiermee heb je alle (on)gemakken van native executables.

GraalVM is beschikbaar als community (CE) en als Enterprise editie (EE) (referentie 10). Voor die laatste heb je wel een licentie van Oracle nodig, maar bevat dan ook de optimalisaties en commerciële features die niet aanwezig zijn in de CE editie.

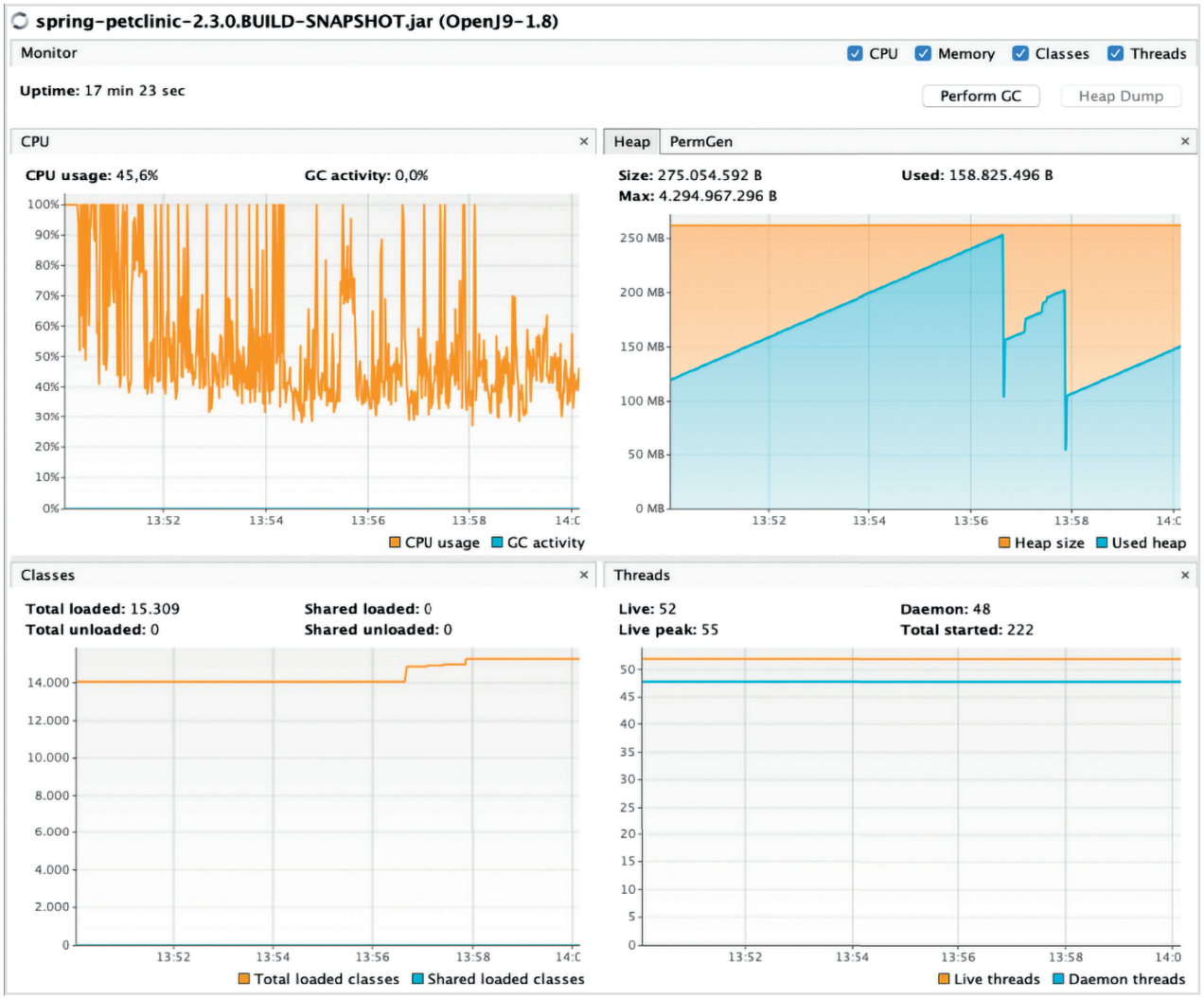
Waarom een andere JVM?

Zoals eerder gezegd verschilt vele JVMs op zijn interne werking, maar ook op eventuele commerciële features die deze biedt of het doel waar het origineel voor gemaakt is. Enkele belangrijke, makkelijk meetbare punten waar je voordelen op kunt baseren zijn hieronder beschreven.

Opstarttijd

Als opstart tijd een hele belangrijke requirement voor jouw applicatie is, dan heb je een uitdaging als je met Java werkt. Hier wordt hard aan gewerkt door JVM's als GraalVM, maar ook OpenJ9. Voor langlopende processen als servers kan het zijn dat dit helemaal geen belangrijke eis is.

ALS OPSTART TIJD EEN HELE BELANGRIJKE REQUIREMENT VOOR JOUW APPLICATIE IS, DAN HEB JE EEN UITDAGING ALS JE MET JAVA WERKT



Afbeelding 1.

CPU en geheugen gebruik

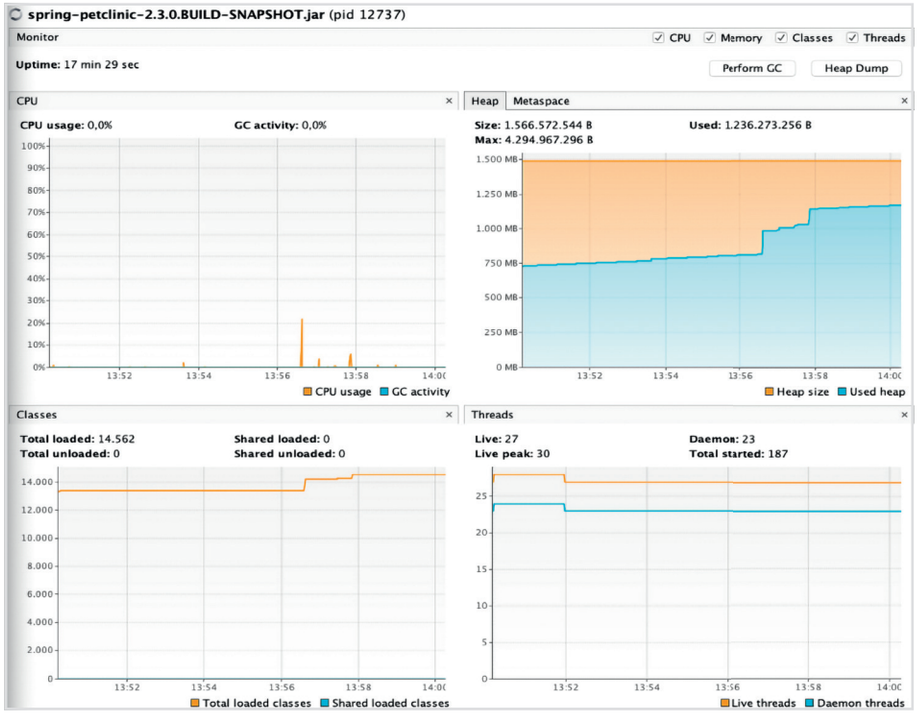
Als je je applicatie in een cloud draait, is het nuttig om te checken hoeveel resources deze gebruikt. Clouds baseren vaak hun rekening voor afnemers op het geheugen en/of CPU gebruik van applicaties die je daar draait. Als je naar een JVM overstapt die minder geheugen en/of CPU gebruikt, kan dit een behoorlijke positieve impact hebben op de rekening die je betaalt.

Om een impressie te geven hoe groot de verschillen kunnen zijn, is hier een voorbeeld van het startup geheugengebruik weergeven gebaseerd op twee verschillende JVM distributies. Hierin wordt de spring-petshop applicatie (referentie 6) gebouwd en gedraaid op twee verschillende JVM distributies: de AdoptOpenJDK Hotspot
JVM 1.8 en de Eclipse OpenJ9 1.8. De applicaties zijn alleen gestart, verder zijn er geen handelingen uitgevoerd. VisualVM (referentie 7) is gebruikt om ze te monitoren.

Afbeelding 1 laat een aantal statistieken van de OpenJ9 1.8 JVM zien. Het geheugengebruik bij startup was rond de 120 MB en loopt daarna snel op naar rond de 250 MB om daarna weer snel door de Garbage Collector (GC) opgeschoond te worden. Het CPU gebruik fluctueert flink maar zit gemiddeld op zo'n 40%.

Het opstart geheugengebruik van de Hotspot 1.8 JVM (afbeelding 2) begint met een geheugengebruik van bijna 730 MB en loopt de tien minuten daarna op naar ongeveer 1200 MB. Het CPU gebruik daarentegen is bijna 0%.

Dit zijn significante verschillen voor JVM's, zonder enige vorm van tuning. Dit is een simpele test en is bedoeld om grote verschillen te



Afbeelding 2.

illustreren. Meer testen zijn nodig, zeker op de instanties voor productie om tot weloverwogen keuzes te komen.

Extended support

In het geval je op een project zit dat zich prima houdt op Java 8 en nog geen plannen heeft te migreren naar een nieuwere versie, is het mogelijk om een distributie te kiezen die doorlopende Java 8 support biedt voor de JVM. Denk daarbij aan dat security-, performance- of stabiliteitsverbeteringen worden gepatcht zonder dat je een nieuwere versie van een JDK hoeft te adopteren. Hier zijn zowel community als commercieel gedreven versies van, zoals getoond in de eerder vermelde lijst. Dit fenomeen kan dus grote impact hebben bij het maken van een keuze.

Specifieke Garbage Collector (GC)

Sommige applicaties, zeker wanneer deze in een cloud draaien, hebben behoefte aan een specifieke GC die het best past bij het soort gedrag van de applicatie. Als een applicatie vaak start/stopt, in tegenstelling tot gestart worden en dan maandenlang draaien, is er zeer waarschijnlijk behoefte aan een GC die die optimaal de resources beheert in de draaiende JVM. Niet alle beschikbare JVM distributies ondersteunen elke GC, dus is het verstandig hiermee rekening te houden bij het zoeken naar de juiste distributie.

Conclusie

Omdat out-of-the-box gedrag per JVM al verschil kan maken, loont het de moeite om verschillende distributies eens uit te proberen om te kijken of je applicatie hierdoor beter gaat performen. Als je een specifieke feature van een JVM nodig hebt, of een support contract hebt, is het verstandig om te inventariseren welke JVM dit contract aanbeveelt of afdwingt. Het kiezen van de juiste distributie kan daarom op verschillende fronten van belang zijn. En laten we eerlijk zijn, als developers onder elkaar: het kan best leuk zijn om te experimenteren met de JVM als “motor” die jouw applicaties als “auto” laat rijden. ■

HET LOONT DE MOEITE OM VERSCHILLENDE DISTRIBUTIES EENS UIT TE PROBEREN OM TE KIJKEN OF JE APPLICATIE HIERDOOR BETER GAAT PERFORMEN

REFERENTIES

[1] **https://www.oracle.com/java/technologies/javase-downloads.html**
[2] **https://jdk.java.net/**
[3] **https://openjdk.java.net/**
[4] **https://adoptopenjdk.net/**
[5] **https://www.eclipse.org/openj9/**
[6] **https://github.com/spring-projects/spring-petclinic**
[7] **https://github.com/oracle/visualvm**
[8] **https://aws.amazon.com/corretto/**
[9] **https://github.com/corretto**
[10] **https://www.graalvm.org/downloads/**
[11] **https://www.graalvm.org/reference-manual/native-image/**