incusdata (pty) ltd. 1995/04149/07
Coding · Matters
PO Box 148
Wierda Park, 0149
tel : +27 12 666 2020
info@incusdata.com
www.incusdata.com

Price: R9,600 excl. VAT
Duration: 3 days
Delivery: Virtual classroom or on-site training

JVM Performance Tuning

Available on request for group bookings only

Description

The JVM Performance Tuning course will help you move beyond an understanding of Java syntax to explore how the JIT compiler manages code execution. You will learn how how garbage collectors manage memory, and how to select appropriate memory sizes and the right garbage collectors. The course covers how modern threading models impact throughput. You will learn the internals of the JVM, memory management and problem diagnostic strategies.

Objectives

After you have completed the JVM Performance Tuning course, you will be able to:

- Understand the role of the interpreter and Just-In-Time (JIT) compilers in the JVM.
- Understand JVM memory management and how JVM garbage collection works.
- Know how to use JVM command line switches to control garbage collection and memory management.
- Know how to measure performance measurement and run diagnostics.
- Understand the concept and cause of the most common memory leaks, and how to find, prevent and fix them.

Intended Audience

You should attend the JVM Performance Tuning course if:

- You are an advanced Java programmer and you want to learn about the internals of the JVM and how to optimise memory usage and garbage collection.
- You are a Java performance engineer and you need to optimise the execution of Java applications.
- You are a DevOps engineer and you need to tune a Java JVM for containerization.

Prerequisites

Before you attend the JVM Performance Tuning course:

- You must have attended our Advanced Java Programming course or
- You should have at least one year of practical experience programming in Java.
- You should have strong proficiency in Java SE programming and some familiarity with Java concurrency basics.

Course Contents

Overview

- JVM internals: The engine under the hood.
- JVM architecture and execution model.
- HotSpot compiler internals.
- Garbage collection (GC) fundamentals.

- GC algorithms and selection strategies.
- JVM memory configuration and tuning.
- Controlling the JVM: Tuning for hardware and efficiency.
- Performance measurement and diagnostics.
- Memory Leaks: Detection, prevention and cures.
- Writing high performance Java code.

JVM Architecture and Execution Model

- Execution engines: interpreter vs JIT.
- Interpreter: Bytecode to machine code interpretation.
- JIT (Just-In-Time) compiler: Compiles to native code for speed.
- HotSpot Compiler: Hot spot detection.
- C1 (client compiler): Moderate optimization, fast compilation (focus on startup).
- C2 (server compiler): Heavy optimization, slower compilation (focus on throughput).
- Tiered Compilation: How the JVM balances C1 and C2 for optimal performance.
- JVM Code Cache.
- JIT logs and diagnostics.

Different JVMs: HotSpot, OpenJ9, GraalVM

- HotSpot vs. OpenJ9.
- Traditional JVM vs. Native Image (GraalVM).
- Ahead-Of-Time vs Just-In-Time Compilation.

Memory Management

- Java heap structure and generations.
- Young Generation: Eden and Survivor spaces.
- Eden Space: for new objects.
- Survivor Spaces (S0 and S1): survivor spaces for aging objects.
- Old/Tenured generation for long-lived objects.

Garbage Collection and Algorithms

- Stop-the-World (STW) events.
- Mark-and-Sweep vs. copying algorithms.
- Incremental GC: minimizing pause times.
- Modern Garbage Collectors: Serial GC, Parallel GC, G1, ZGC, Shenandoah.

JVM Tuning and Command Line Switches

- Memory Sizing: heap and generation sizes
- Metaspace and class metadata tuning.
- Stack sizing per thread.
- Selecting the right garbage collector: Latency vs. throughput.

- Tuning for pause time goals.
- Controlling JIT performance: compilation thresholds.
- Printing compilation details.
- On-Stack Replacement (OSR).

Diagnosing and Fixing Memory Leaks

- Understanding memory leaks in Java.
- Common causes: static collections, unclosed resources, inner class references, improper ThreadLocal clean-up.
- Tools for leak detection: jmap, jstat, jcmd, jconsole.
- Analysing heap dumps.
- Profiling tools: VisualVM, Java Flight Recorder.
- Prevention strategies and best practices.
- Using weak/soft references.
- Proper lifecycle management and the try-with-resources pattern.

Advanced Threading and Virtual Threads

- Traditional platform threads.
- Project Loom and virtual threads.
- Pinned virtual threads.
- Thread pool tuning.

Writing High Performance Java Code

- Algorithmic complexity vs real-world performance.
- Choosing appropriate data structures.
- Avoiding unnecessary object creation.
- Object pooling: Pros and cons.
- Data locality and the role of the CPU cache.
- The cost of boxing/unboxing.
- Benchmarking: Using JMH (Java Microbenchmark Harness) to test performance.

** *The lecturer reserves the right to modify the contents of the course to suit the needs of the delegates.*