Maven Dependency Analysis using Akka Streams

Nikita Servais

November 26, 2023

1 Introduction

This report outlines the design decisions and implementation approach for the Maven Dependency Analysis project, accomplished using Akka Streams and Scala. The project's objective is to process library dependencies from the Maven repository, focusing on analyzing the number of compile and runtime dependencies for each library.

2 Design Decisions

2.1 Akka Streams for Concurrent Processing

Akka Streams were chosen as the foundation for this project due to their robust framework for building streaming applications with built-in support for backpressure. This ensures efficient resource utilization without overwhelming system components, which is vital for processing a large dataset of Maven library dependencies.

2.2 Scala for Functional and Object-Oriented Paradigms

Scala offers a seamless integration of functional and object-oriented programming paradigms, ideal for creating immutable data structures and employing functional transformations. This aligns well with Akka Streams, which is built on top of Scala and Akka actors.

2.3 Data Model Design

The core data model includes case classes *MavenDependency, Library, Dependency,* and *MavenLibrary*, representing the library, its dependencies, and their types. Scala's case classes provide immutability and automatic getters, equals, hashCode, and toString methods.

3 Akka Stream Components

3.1 Source

FileIO.fromPath is used for reading the CSV file, offering non-blocking IO operations essential for stream processing.

3.2 Flow Components

- **parseFile:** Parses the CSV file, using `Framing.delimiter` for efficient line handling.
- **instantiateClasses:** Transforms CSV lines into *MavenDependency* objects.
- **groupByLibraryName:** Groups dependencies by library, showcasing stateful streaming operations.
- **throttleGroups:** Throttles processing to 10 groups per second, managing backpressure.
- **bufferGroups:** Buffers the stream with a capacity of 5 elements, handling data flow under load.
- **countDependenciesFlow:** A custom flow created using GraphDSL to count dependencies in parallel.

3.3 Sink

Sink.foreach is used for outputting the final result, demonstrating the role of a sink in stream processing.

4 GraphDSL for Custom Flows

GraphDSL allows the creation of complex stream processing topologies, used here for the *countDependenciesFlow*.

5 Throttling and Buffering Strategies

Throttling and buffering are employed to manage the data flow, ensuring system responsiveness and stability.

6 Conclusion

The Maven Dependency Analysis project demonstrates the capabilities of Akka Streams in processing large datasets. By leveraging Scala and Akka Streams, the application efficiently and effectively analyzes Maven library dependencies.