# Docker: Go, DockAlt: Java, OCaml, Scala Implementation

Patrick Shih (UID: 604580648), Discussion 1B

## Abstract

Docker is a software construction platform that is built on the Linux containers. Docker's purpose is to provide a standard development environment for developers, and to allow sharing and collaborating applications. The issue with Docker comes with its young history and single source, in which if Docker ends up buggy, it would be no longer usable. This report looks into the possibility of implementing Docker using different programming language, called DockAlt. Docker uses the programming language Go, which will be compared to DockAlt's implementation with one of the following three programming languages: Java, OCaml, and Scala.

## Introduction

Containers are used to virtualize an operating system, so that multiple applications can be ran at the same time on one operating system. To compare to VMs, the analogy is that rather than using 100 computers to address 100 applications, containers find a way to use one single computer to address 100 applications. In this report, we look to see what programming languages would be best to implement a container, DockAlt.

## 1. Go

Go is a rather novel language with many different features. Go has a static compilation, supports concurrent programming, contains a garbage collector, duck typing, and an extensive library. Go attempts to stray away from object-oriented programming by not including aspects such as inheritance and pointer arithmetic.

### 1.1. Advantages

Go seems quite compatible with Docker's specification: its static compilation, asynchronous processing, low-level interface, and designer-friendly qualities are all advantages of Go as Docker's programming language.

#### 1.1.1 Static Compilation

A powerful advantage of Go is its static compilation. When a program finish compiling in Go, all the library files and references that the program used are no longer needed. The relevant parts of libraries are in the executable; therefore, the libraries do not have to be saved. [1] This is important when implementing containers like Docker, as the main objective of containers is to utilize as little space as possible. Static compilation does such task by not having an entire library saved when needed to be referenced.

#### 1.1.2 Asynchronous Processing

Go supports asynchronous processing, which is when threads do not need to wait on other threads for completion. Asynchronous processing is therefore incredibly important for containers, as some programs may require containers to deal with a lot of threads. [2] If threads are blocked by other threads before, this would potentially exhaust memory and accesses to the container. Asynchronous processing handles and reduces the waiting, so threads do not have to stay idle.

#### 1.1.3 Low-level interface

Go has a low-level interface that is important for Docker to use. A low-level interface provides detailed and exact functions and modules that lead to low-level access. Low-level access takes a minimal effort to throw code into the hardware and ignore the software transition layer, thereby increasing speed. [3] As stated earlier, Docker attempts to maintain a high-performing and quick platform, and a low-level interface would produce better results in comparison to a high-level interface.

#### 1.1.4 Duck typing, library, development environment

Go is a statically typed programming language that supports duck typing, known as structural typing. In essence, this takes away inheritance that is found in languages like C++ and Java, but still support "inheritance qualities". This is supported by Go's usage of deriving types that corresponds with interfaces [4]. Go also has a full development environment, where developers can find open-source code and documentation for certain resources. Finally, Go's library is extensive and available to users. All three of these qualities are important on the developer side of Docker, as its platform would not only be easier to maintain, but it would also be easily understood or researched.

### 1.2. Disadvantages

There are a few disadvantages worth noting in regards to Go, such as concurrency issues, change management, and error-handling. [5]

#### 1.2.1 Concurrency

Go uses maps that are not thread-safe. This issue is not fully addressed by the Go, rather they argue that thread safety should be handled by the developers, not the programming language. Suggestions are using sync.Mutex and other basic access protections. On the flipside, because maps are not thread-safe, they are fast. However, if a designer is not careful with keeping threads safe, this would incredibly ruin performance.

#### 1.2.2 Change Management

Go's change management has some flaws: 'go get' cannot pin revisions, 'go test' do not have deconstructors, and 'go build' raises difficulties when running packages with similar code. Change management is important for developers to maintain and test code among a team. Developers at Docker would have to consistently address these issues – possibly leading to team managing problems.

#### 1.2.3 Error Checking

Go's error handling is not the most sophisticated, as it could potentially be verbose. This is an important aspect to note on the developer side of Go, as the need for test quality assurance is important for any products. However, this issue is more styling than of anything problematic with Go.

## 2. Implementing DockAlt

Upon developing DockAlt, there are two implementations to consider – utilize the same qualities from Go or use a different quality and compare it to Docker. That being said, the advantages and disadvantages are a bit more difficult to quantitate without a prototype or deep knowledge of a language. Therefore, for Java, OCaml and Scala, I will be highlighting the five main technical challenges of implementing DockAlt.

## 3. Java

Java is a general programming language, with an extensive history as one of the strongest and most relevant object oriented programming languages.

### 3.1 Ease of Use

Java has a better usability than Go, but that may simply be due to its longevity and constant updating. Java is an incredibly popular language, which would be one of the main positive attributes to implementing DockAlt with it. Any learning curve or understanding syntax and libraries can be easily addressed. Java is also well documented through Oracle. Java also has a simple and usable IDE in Eclipse and it also has an extensive library from the Java Class Library. Finally, Java's error handling is well developed. Java's exceptions and try/catch would aid in test cases and quality assurance.

### 3.2 Flexibility

Java is a generic programming language, but with an emphasis on object-oriented programming. Java is indeed powerful, as it has multiple libraries for different programming purposes, such as scripting, networking, and concurrency. This flexibility allows a lot of documentation and existing qualities that could aid in implementing DockAlt.

### 3.3 Generality

Java is used industry-wide for many services, especially with game development, embedded systems, and parallel computing. All of these show Java as a high performance language that is incredibly malleable in usage.

### 3.4 Performance

While Java's performance in general is well built, it is important to view Java's performance in context of implementing DockAlt. Java is also an interpreted language, which raises the question of speed as a high-level interface. Java addresses this performance issue with their just-in-time compiler, which compiles bytecode into machine code at run-time. Because of this attribute, developing DockAlt can be possible without much of a cost in performance. However, with the process of interpreting is still relevant. A prototype benchmarking speed can help aid in analysis of performance.

### 3.5 Reliability

While Java is reputable for their robustness, a possible issue could arise in its garbage collection. Java's memory management only collects garbage when the collector is full. [6] When implementing DockAlt, there could raise a possibility that there would be a lot of information being sent in to the container. Without constantly updating and looking for space, the process may affect performance. Dealing with this problem with Java memory management may lead to memory leaks.

## 4. OCaml

OCaml can be considered both functional and object-oriented programming language.

### 4.1 Ease of Use

While OCaml is well documented and has a solid library, perhaps the most difficult part of implementing DockAlt would be the mastery of functional programming. OCaml's pattern matching and tail recursion may not be the most intuitive when writing a program. However, OCaml does not have the same verbose syntax that Java has. OCaml is reputable as concise and fast language.

### 4.2 Flexibility

OCaml's main focus attribute lie in its background as machine learning derived language. Later built were functional and object-oriented programming qualities. OCaml does not necessarily display the same types of flexibility that Java does in terms of its usage.

**4.3 Generality**

Looking at different softwares use OCaml for what purposes examines how general the environment can be for the language. For example, MLDonkey uses OCaml as a peer to peer client. Among others, OCaml can be written for compilers, package manager, software library, and file synchronization [7]. What is important to note is that there were no examples of containers or even virtual memory for OCaml, but there are certain examples that show potential for DockAlt.

**4.4 Performance**

Similar to Java, OCaml is an interpreted language that uses bytecode to machine language compiler at runtime. OCaml is also reliant on tail-recursion, which changes recursive calls into loops and saves time. [8] OCaml also has type inference, which means that it determines type during run-time so the compiler does not have to exert resources, unlike Java. To put it short, OCaml is built to be fast, a very much needed quality when implementing DockAlt.

**4.5 Reliability**

OCaml's type inference may possibly lead to difficulties in debugging and implementing code. From a beginner's perspective, OCaml's typing errors may be difficult to interpret or handle. Also, OCaml, and all functional languages, is reliant on recursion. Failing to implement a tail-recursive algorithm would be very space-dependent. However, these issues are all addressed by the experience of the programmer. An inexperienced developer using OCaml for DockAlt would raise the possibility of using too much memory – a clear detriment to DockAlt's platform.

## 5. Scala

Scala is short for "scalable languages," and takes qualities of object oriented programming and fuses it with functional programming.

**5.1 Ease of Use**

Scala runs on Java's Virtual Machine, so if familiar with Java in general, especially its framework and documentation, Scala would be an easy pick up as a programming language. A key feature of Scala is that Scala can also interpret Java code.[9] This is important, as writing DockAlt with a language based on Java would also have similar documentation that one could easily research.

**5.2 Flexibility**

As stated earlier, Scala is both object-oriented and functional. Scala is also used for parallel and concurrent programming, along with metaprogramming. Its flexibility is notable for implementing DockAlt, as functional programming is often related with speed, and concurrent programming is an important attribute of DockAlt.

**5.3 Geniality**

A lot of Scala applications are based on social networking websites and in general software engineering. [10] As a growing language, we can expect Scala's usage to expand.

**5.4 Performance**

The main notable quality of Scala is its typing inference. Like in OCaml, typing would be done at runtime which would speed up the process. Scala takes features like tail-recursion and pattern matching, both processes from functional programming that is attributed for its speed, and combined with the extension and inheritance from object oriented programming. Similarly, like OCaml and Java, Scala uses Java's bytecode compiler. This will also affect speed. While the speed is not as fast as a pure functional program, for DockAlt, it could be good enough.

**5.5 Reliability**

Because Scala takes many of the same features from Java along with functional programming, there have been reports of how the two types of language clashes with each other. For example, when attempting to create a very large library with tuples, issues occur of representing such structure.[11] While this issue was eventually fixed with a patch, it is worth noting how Scala deals with large data structures, as an implementation for DockAlt would indeed involve such.

## Conclusion

All three languages have their pros and cons. All of them may expect slower performances due to their reliance on a compiler. However, if speed is the priority, OCaml seems like a likely candidate, as most of the cons are based more on user experience rather than internal issues of the language. Scala is also a proper choice to implement DockAct, as it takes a lot of positive features of Java and addresses its speed issues with functional programming.

## References

[1] https://en.wikipedia.org/wiki/Static_build

[2]https://books.google.com/books?id=xSGKAwAAQBAJ&pg=PA372&lpg=PA372&dq=asynchronous+proc

essing+containers&source=bl&ots=77_SB2I_2X&sig=xuu5-4FMyyluGlxKJvYBUQFr8Bw&hl=en&sa=X&ved=0ahUKEwjQqpGJ9t7SAhVE4GMKHeZcCu0Q6AEIQDAG#v=onepage&q=asynchronous%20processing%20containers&f=false

[3]http://www.pcmag.com/encyclopedia/term/46354/low-level-access

[4]http://www.club.cc.cmu.edu/~cmccabe/blog_golang_type_system.html

[5] https://www.slideshare.net/jpetazzo/docker-and-go-why-did-we-decide-to-write-docker-in-go/36-cant_select_on_readerswriters_for

[6] https://docs.oracle.com/cd/E13150_01/jrockit_jvm/jrockit/geninfo/diagnos/garbage_collect.html

[7] https://en.wikipedia.org/wiki/OCaml

[8] https://ocaml.org/learn/tutorials/performance_and_profiling.html

[9] https://www.scala-lang.org/what-is-scala.html

[10] http://www.scala-lang.org/old/node/1658

[11] https://www.forbes.com/sites/quora/2014/07/10/what-do-you-think-about-the-scala-programming-language/#49f71c132064