DockAlt: Alternative Containerization Implementations and Semantics

Eric Sehun Oh
University of California, Los Angeles
Computer Science 131, Fall 2016

Abstract

Docker is an open-source project that seeks to solve the 'matrix of hell' problem, the challenge of making an ever increasing set of applications, languages, frameworks, etc to interact properly with an ever increasing set of hardware environments, by automating the deployment of Linux applications inside software containers. DockAlt seeks to explore alternative implementations to Docker in languages such as Java, OCaml, and Dart to Docker, which is currently implemented in Go.

1 Introduction

LXC is a userspace interface, providing a virtual environment that has its own process and network space, allowing users to easily create and manage system or application containers. It is seen as a middle-ground between a chroot and a virtual machine as it lacks the need for a separate kernel as it shares the kernel with the host and packages only the needed applications rather than an entire operating system and hardware setup like a virtual machine would.

1.1 Docker

Docker is a project by dotCloud written in Go that was initially based on the LXC project to build a single application container. Although not a replacement for LXC, Docker offers additional high-level tools on top of LXC's low-level foundation of kernel features. In order to enable portable deployment across different machines, Docker provides an abstraction to deal with the 'matrix of hell' problem by standardizing containers, allowing developers and operators to be less concerned with the underlying infrastructure. By defining a format in which applications and dependencies are bundled, Docker allows for a single object to be transferred to other Docker-enabled execution environments in which the object will exhibit the same execution semantics.

1.2 Docker Implementation in Go

Docker's language choice for implementation is Go as it offers a few key benefits. Go offers static compilation which allows for ease of installation, testing, and adaptation. It is a good candidate for bootstrapping with no dependencies and thus enables generated executables to run in new environments fairly quickly. Additionally, it offers good asynchronous primitives, low-level interfaces (managing processes and system calls), extensive standard library and data types, and strong duck typing, which is a type of dynamic typing in which type checking is deferred to runtime. Because Go does not use preprocessors or includes and thus is one reason its compilation time is very quick.

With tradeoffs imminent in any implementation, there are a few drawbacks that come with Go. Maps in Go are not thread-safe although they are very fast in nature. Additional precautions must be taken into account by protecting access with a sync.Mutex or with the use of channels. Also, Go's error handling can also be verbose, similar to the syntax itself.

2 Java

Java is one of the most popular, object-oriented programming languages that is similar to Go in ways regarding garbage collection and concurrency framework. Java's concurrency is however different in that its mechanism natively synchronizes access to memory while Go handles such problems of synchronization by communicating over channels in which reads and writes occur. These synchronization tools for Java, however, is dependent on libraries while as Go can accomplish most tasks with its standard libraries.

Java is known to be very portable as one can run Java bytecode on any hardware that has a compliant JVM, a Java Virtual Machine. Java shares a similar idea with Docker in that one program should be able to run on any computer with JVM/Docker installed.

Since Java is so portable, with this regard it would make it a very viable substitute for Go in Docker.

Error handling is an area in which Java differs Go. Go's approach to communicate errors via separate return value contrasts with the commonly used concept of exceptions in Java. Another difference is compilation; Java is just-in-time compilation meaning that it is dynamically compiled, with compilation done during the execution of the program. This can ultimately be detrimental when the compiler must figure out dependencies as the compiler must also install them.

3 OCaml

OCaml is a functional programming language that combines concepts of imperative and object-oriented programming. OCaml's main strength lies in its performance as it is statically typed, rendering runtime mismatches unlikely, guaranteeing runtime safety. It is much like Go in a sense that it has garbage collection for automatic memory management, a feature found often in modern high-level languages, and is natively compiled producing performant code without requiring heavy optimization or containing complexities that come with dynamic compilation such as JIT, compilation found in Java.

A key difference between OCaml and Go is the lack of proper concurrency and multicore support, and thus has to rely on on external libraries such as LWT or Async to achieve the same kind of things that Go can do in its built-in libraries and routines called goroutines.

4 Dart

Dart is a general-purpose programming language that was developed by Google and is often used to build web, server, and mobile applications. Its syntax is very similar to C and Java and supports similar features such as object-oriented programming and concurrency. Dart also includes a comprehensive core library with features to aid with: async, collections, regexps, typed data, and etc. This remove the unnecessary need to include external resources for basic functionalities.

Dart has a few main ways to run, one of which is compiled as JavaScript. Dart relies on a source-to-source compiler to JavaScript, transcompiling source code to optimized JavaScript code, often written more quickly than using native JavaScript.

5 Implementation Comparisons

5.1 Ease of use

Duck typing is a feature which allows for great ease of use. Regarding this feature, the languages that support it are Go, OCaml to an extent, and Dart. Java does not support it and take a hit in ease of use for our DockAlt implementation, despite its widespread use and syntactic similarity.

5.2 Flexibility

Error handling in Java is with exceptions and is handled individually. However, when it comes to flexibility Go's approach with multiple return values allows for a state in which the program can continue even when there is an error. This can offer new ways of implementation as long as it is handled well by the developer.

5.3 Generality

Generality is one area in which Dart excels as its strong core library offers many functionalities without the need for external libraries and thus much of the code will be very similar and simple. OCaml albeit great for performance and high-level programming, is not for one in terms of generality.

5.4 Performance

In terms of performance, OCaml and Go are very similar with Dart following closely behind. Both also follow a straightforward, native compilation strategy that produces performant code without requiring heavy optimization. Java falls short as it is an interpreted language which dramatically slows down performance and is bogged down by additional complexities in JIT compilation. OCaml and Go are probably the most efficient, generating fast code running with a small memory footprint.

5.5 Reliability

Regarding reliability, OCaml has great strength it safety because it verifies at compile time that your program is safe, correct apart algorithmic errors. Once it compiles you are almost sure that I will work. Go and Java does not give this guarantee, as in Java there are a number of runtime errors which occur from time to time.

5.6 DockAlt choice

Overall, the best choice for an alternate implementation would have to come down to Dart or OCaml. Java is inherently very slow due to its dynamic compilation and is not what we need for DockAlt. Dart offers very good async support while OCaml will need external libraries and dependencies. OCaml's strength lies with its security and reliability along with its speed.

References

[1] Golub, Ben. *PaaS: Present And Future.* 16 Aug 2013. https://blog.docker.com/2013/08/paas-present-and-future/. 29 Nov 2016

[2] Williams, Alex. *The Matrix Of Hell And Two Open-Source Projects For The Emerging Agnostic Cloud.*28 Jul 2013. https://techcrunch.com/2013/07/28/the-matrix-of-hell-and-two-open-source-projects-for-the-emerging-agnostic-cloud/. 29 Nov 2016

[3] Banerjee, Tobby. *Understanding the key differences between LXC and Docker.* 19 Aug 2014. https://www.flockport.com/lxc-vs-docker/. 29 Nov 2016.

[4] Schwartz, Ben. *Concurrency in Java and Go.* 09 Apr 2014. http://txt.fliglio.com/2014/04/concurrency -in-java-and-go/. 01 Dec 2016.

[5] Baukes, Mike. Docker vs LXC. https://www.upguard.com/articles/docker-vs-lxc. 01 Dec 2016