CS 131 Homework 6

Sinan Cem Cetin

Abstract

Docker is an open source containerization platform built on top of Linux Containers (LXC) to create standardized applications through running them on multiple environments on the same machine in a manner that is both lightweight and fast. Since Docker is a relatively new platform and there is only one source for it, it is easy to be apprehensive when it comes to implementing it. The aim of this study is to find alternatives for Docker using languages such as Ocaml, Java and Scala and compare the languages to Go, the language Docker was originally written in to build DockAlt.

Introduction

Containerization is the ability to package up dependencies, namespaces, cgroups and other application specific libraries into a single container as an image. This image is then able to be moved around, allowing for the development and deployment onto different types of containers to test and debug. As this is what Linux Containers do for the Linux kernel, Docker was built on top of LXC using bindings and the additional abstraction provided by the libcontainer. Docker allows for one host running an application on a particular OS to run the application in different containers which are faster and more lightweight than Virtual Machines. In this study, I svaluate the choice to build Docker in the language Go compared to other Languages such as Ocaml, Java and Scala.

Go

Go is a relatively new language released by Google in 2009. One of its defining strengths is its support for concurrency using goroutines, which are modularized functions that are automatically distributed amongst threads and channels, which are piping of concurrent goroutines that make the code thread safe. Go is statically compiled, which means that it compiles to machine code before it is executed. This is advantageous as it requires to extra installations to run and it can use popular dynamic libraries that are on many systems. Since Docker needs ease of transition between different systems and platforms, Go is particularly useful. Go is also platform independent due to its ability to build for multiple platforms and this allows for the elimination of extra dependencies making it ideal for the bootstrapping employed in Docker containers. Go also features an official LXC binding, allowing for good low-level interfaces to manage processes, system calls etc. Go's strong duck typing also allows for the dynamic type checking to work as type inference, making development for Docker easier. Go also provides a full development environment upon installation which allows for easy access to documentation, dependencies on remote servers etc making the development process more user friendly. The only real disadvantages to using Go is that goroutines and channels will sometimes sacrifice safety for speed and that Go is a relatively new language. [1]

OCaml

OCaml is a strong statically-typed language with type inference that is done with compile-time bindingsWriting OCaml code is particularly fast as the developer does not need to watch and predetermine types of the data, but just needs to ensure the operations on the data is consistent. Failure to do so leads to quick error detection during compile time. This could make developing DockAlt faster. OCaml is compiled down to bytecode which could be made specific to the system or to an interpreter that comes with the OCaml toolchain. This would be useful for an application like DockAlt where the code really benefits from being portable and machine agnostic. OCaml also comes with powerful libraries that allows for good low level interfaces such as the unix library or the sys library.

The biggest disadvantage to using OCaml is that it could take a while to get used to the functional paradigm of thinking as OCaml is very functional in nature, it would be hard for programmers with no prior experience with the language to get started working on DockAlt. One of the other main issues is that there is no officially supported library for binding to LXC, this means that creating the containerization support would be a more difficult task than using languages with official support such as Go. The final issue is that OCaml requires a Just-In-Time compiler or a bytecode interpreter to be associated in the build package for DockAlt, otherwise it would require extra dependencies.

Java

Java is an older language that focuses on the object-oriented paradigm of programming. Java uses static type checking which could be cumbersome while writing code for the program. However, the static typing would allow for errors to be caught earlier on, during compilation rather than in the runtime interpreter stage making DockAlt easier to debug. Java is very portable, as it, like OCaml gets compiled down to byte code that can be run on the JVM regardless of the machine architecture being used eliminating a need for multiple build versions as the JVM would take care of most of the low-level machine-specific worries. Java's age also means it has a strong community and a huge library for external modules to be integrated into the application. This could come as both an advantage and a disadvantage to DockAlt development.

Because Java is built to be platform agnostic, it has no default binding to LXC. DockAlt also requires the ability to interface well with the extremely low-level interfaces of any operating system. This might be difficult without incorporating external libraries into the application. Also the JVM is not on every machine by default, without the JVM coming with the DockAlt system, the Java bytecode would be pretty useless. DockAlt would have to be further bloated to allow for the Java code to run. While statically typed languages make the program safer, they make the code more complex which lengthens the development process for DockAlt.

Scala

Scala is a functional and object-oriented language that also runs on the JVM. It has the same benefits of Java such as Java code's inter-platform operability and like Java, Scala also has strong static typing so more errors can be caught at compile time, making the code more reliable and easier to debug. Developers who are familiar with functional programming would be able to adapt to working with Scala easily. Syntactically, Scala shares similarities with OCaml and it features type inference, function currying and immutable data. Scala's functional nature means the code is free from side effects compared to Java and avoids some of the shortcomings and edge cases of that languages.

Scala suffers from the same main disadvantage as Java, that it has no native LXC binding making it difficult to implement an application like DockAlt that relies on interfacing with LXC. Another issue with Scala is that, like in OCaml, its functional programming might have a steep learning curve for programmers who have never worked with functional programming languages before.

Conclusion

Of the three programming languages examined in this study, the two that seem to be the most suited to implementing DockAlt is Go and OCaml. Despite the smaller community, OCaml's straightforward interfacing with C code would work with a project like DockAlt which would rely heavily on interfacing with LXC which has a C-based API. It's functional nature and type inference also makes OCaml a favorable candidate for the development of DockAlt.

Go, on the other hand, as explained by the devs of Docker, is a language worth giving a shot. While relatively new, Go provides good low level interface while having better memory safety than languages like C. However Go's newness also shows in the lack of features for testing and packaging as well as tools that older languages like Java and Scala have had for a long time.

References

1. Petazzoni, Jérôme. "What's a Linux container? High." LinkedIn SlideShare, 7 Nov. 2013, www.slideshare.net/jpetazzo/docker-and-go-why-did-we-decide-to-write-docker-in-go/11-Whats_a_Linux_containerHigh_level.