The feasibility of implementing DockAlt with Java, OCaml and Rust

Jingyue Shen, 704797256

1. Abstract

This report discusses about the possibility of implementing an alternative for Docker called DockAlt. We mainly focus on three languages implementation: Java, OCaml and Rust to see their pros and cons.

2.Introduction

2.1 Linux Container

Linux Container is a lightweight virtual environment that groups and isolates a set of processes and resources such as memory, CPU, disk, etc., from the host and any other containers. Unlike VM, Linux Container shares the same kernel with the host and thus with much less overhead. The isolation guarantees that any processes inside the container cannot see any processes or resources outside the container. And the layered file system makes it easy to move around. [1] It can be used for deploying and testing applications for production.

2.2 Docker

Docker is an application container that is designed to package and run a single service, unlike OS containers such as LXC which is designed to run multiple processes and services.[2]. It is a runtime for Linux container. It aims to solve the problem of managing multiple components of an application(e.g. database server, application server, frontend app), each of which must be run across multiple environments(e.g. developers' local machines, QA servers, production). In order to enable portable deployment across different machines, Docker provides an abstraction to deal with 'matrix of hell' problem by giving a standardized container. Each component of an application can be run on a separate container and data can be shared among them to achieve separation of concerns.

3. Implementation of Docker (Go)

3.1 Advantage

Docker is implanted using Go. It is a relatively new language that features static compilation, garbage collection, and strong static typing. From the Docker: an insider view[3], the developer chose Go to implement Dockers from several reasons.

First, Go's static typing make it easy to install on a variety of operating systems. There is no dependency required.

Second, it has good asynchronous primitives and concurrency support. Since Go 1.1, a new scheduler comes into play which dramatically increases the performance of paralleled Go programs. It is optimized just for Go and has less overhead than the OS's thread scheduler[4].Since dockers can run different part of a application in different containers and all of them share the same kernel with the host, concurrency and synchronization is really important.

Besides, Go's provides low-level interfaces to directly communicate with the kernel, which is necessary for Ducker.

Also, Go uses "duck typing"(or technically called 'structural typing' since it happens at compile time). If a struct in Go has some methods that match an interface, then programmers can use it whenever that interface is needed without defining that this type implement that interface[5]. This provides more flexibility to programmers.

Since Go does not use any pre-processors, its compilation time is very fast.

3.2 Disadvantages

With tradeoffs imminent in any implementation, there are a few drawbacks in Go. Go's map is not thread safe for the sake of speed. Programmers need to take precautions to avoid race conditions. Also, its error handling is very verbose and Go's package manager lacks the ability to pin libraries to a specific version which impedes reproducible, deterministic builds.

4.Implement DockAlt in Java

4.1 Advantages

Java uses a strong, static type systems, which means that all the type errors will be detected at compile time, and those unwanted implicit type conversions will be warned, which is good for the stability of DockAlt.

Besides, Java runs on Java Virtual Machine(JVM), which enables cross-platform portability for DockAlt. I think it is a great shining point for Java and this is what Docker currently lacks. Docker has different versions for Mac, Windows and Linux.

Also, Java has many well-developed IDEs such as IntelliJ and NetBeans, which are more convenient for writing the DockAlt application.

What's more, Java has numerous different classes for concurrency support. But Java's concurrency is different from Go in that its mechanism natively synchronizes access to memory while Go handles such problems of synchronization by communicating over channels in which read and writes occur.

4.2 Disadvantages

Unfortunately, Java lacks of support for LXC. Though there is third-party implementations, they rely on executing LXC commands on LXC command line interface and parsing the text output back into Java. It is not based on binding of LXC C library to Java[6]. So if we use Java to implement DockAlt, it will be inefficient and inconvenient since DockAlt relies heavily on functionalities of LXC.

Besides, Java is overall the most verbose language of the given options. It's type system is very complicated and has deep inheritance hierarchy that has some little messy cases all around Java standard libraries that do not obey the hierarchical inheritance-driven model[7].

Also, from my research, I believe there is no low-level interface to interact with C functions except the Java Native Interface, which is fairly complicated in that it requires programmers to first write C code and compile it into libraries and then let java to link and load it. Since DockAlt needs to frequently interact with the host kernel, this is very inconvenient.

5 Implement DockAlt in OCaml

5.1 Advantages

OCaml is a primarily functional language with support for object-oriented design along with imperative-style code. It uses a very strong static typing with type inference, which can catch all type-related error and makes the code far more reliable without being as verbose as Java.

Also, OCaml has its own foreign function interface like Ctypes which let you define C interface in pure OCaml without writing C code[8]. This is much better and efficient than Java's approach to do low-level operation and interact with kernel.

5.2 Disadvantages

Though OCaml has third-party implementations already exists that provide bindings to LXC API, but unlike Go, it is not actively developed and tested, which might lead to reliability issues.

Besides, parallelism is a big issue in OCaml. OCaml's built-in thread library has poor performance for both concurrency and parallelism. It is not as lightweight as Java's and only a single thread executes at a time[9]. There are external libraries like Lwt and Async that are well suited for concurrency but will not help for parallelism due to OCaml's non-reentrant runtime. There are many libraries developed to take advantage of multicore computers, but thread parallelism generally does not work in OCaml.

6. Implement DockAlt in Rust

6.1 Advantages

Rust is a systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety[10]. It has foreign function interfaces such as the library libc that provides native bindings to the types and functions commonly found on various systems. So it should interact with the kernel very fast and efficient, which is very suitable for implementing DockAlt.

Also, Rust uses static typing and type inference like OCaml, which can eliminate all type related errors at compile time to provide a more reliable DockAlt platform. And the static typing makes implementation process easier.

Besides, there are several mechanisms to ensure memory safety. The unique ownership system ensures that there is only one binding to any given resources and when a binding goes out of scope, Rust will free the bound resources. This eliminate the risk of dangling pointers since only one variable can take ownership of a resource allocated on the heap. The other one is that the reference type is used to borrow ownership and it is by default immutable. In order to change the binding of a variable, they need to be explicitly declared as "mut". Together with the rule that only one of the following two ownership borrowing can exist at the same time:

- one or more references(&T) to a resource
- exactly one mutable reference(&mut T)

, **Rust is able to prevent data races at compile time**(since only one binding can take ownership and if we use reference to borrow bindings to let other variables bind to the resources, the above rules can avoid simultaneous reading and writing)[11].

Rust does not have a garbage collector. It would know when variables get out of scope at compile time and thus insert corresponding LLVM/assembly instructions to free the memory[12]. While ensuring memory safety, this approach can let DockAlt runs more efficiently since there is no periodic garbage collection compared to Go and Java and OCaml.

For concurrency, Rust has standard libraries to implement this feature like std::thread and its two traits "Send" and "Sync" make strong guarantee about the code under concurrency.

In short, Rust focuses on speed and safety and it is low-level enough to efficiently communicate with kernel and does well on concurrency and synchronization. It's abstractions like ownership system takes zero cost.

6.2 Disadvantages

I think the greatest disadvantage is that it the learning curve of Rust's various zero-cost abstractions is high. It is very complex in order to ensure memory safety at compile time. And given that the language is very new(Rust 1.0, the first stable release, was released on 2015), there might not be enough experts and supports to develop DockAlt. The cost will be relatively higher, both for development and maintenance.

7. Conclusion

In this report, I examined the pros and cons of using Java, OCaml and Rust to implement an alternative for Docker, as well as Go, the language that implements Docker. Of the three languages examined, I believe Rust should be the best choice to implement DockAlt. It is extremely fast while ensure memory safety and no thread race condition at compile time. And it's syntax is somewhat similar to C++. While OCaml is a good option for DockAlt, Rust gives me a sense that it inherits all the great features of OCaml and makes one step further. Though Rust is pretty new, its community is growing very fast. It won first place for "most loved programming language" in the Stack Overflow Developer Survey in 2016 and 2017[13]. So I believe it is a great choice for implementing DockAlt, and is very promising in the future.

8.References

[1] https://blog.risingstack.com/operating-system-containers-vs-application-containers/

[2] https://blog.risingstack.com/operating-system-containers-vs-application-containers/

[3]https://www.slideshare.net/jpetazzo/docker-and-go-why-did-we-decide-to-write-docker-in-go/16-Runtime_for_Linux_containersjpetazzotarrasque_sudo

[4] https://morsmachine.dk/go-scheduler

[5] https://medium.com/@matryer/golang-advent-calendar-day-one-duck-typing-a513aaed544d

[6] https://github.com/waseemh/lxc-java

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

[8]https://realworldocaml.org/v1/en/html/foreign-function-interface.html

[9]https://stackoverflow.com/questions/16562394/what-libraries-should-i-use-for-better-ocaml-threading

[10] https://www.rust-lang.org/en-US/

[11] https://doc.rust-lang.org/book/first-edition/references-and-borrowing.html

[12]https://stackoverflow.com/questions/32677420/what-does-rust-have-instead-of-a-garbage-collector

[13]https://en.wikipedia.org/wiki/Rust_(programming_language)