Containerization Support Languages Report

CS131
Bingxin Zhu
UID: 704845969

Abstract

We would like to run previous project on a large amount of virtual machines by using operating system level virtualization with linux containers to keep the cost low. Docker is a platform that allows for software to be managed and distributed through the use of lightweight standardized Linux containers. We would like to take at look at DockAlt which is currently implemented in Go and seeks to explore alternative implementations to Docker in languages such as Java, OCaml, and Scala to Docker,
Introduction

1. Docker

1.1 Linux Container

Application containerization is an OS-level virtualization method used to deploy and run distributed applications without launching an entire virtual machine (VM) for each app. [1]
Linux Container is a userspace interface for the Linux kernel containment features. Through a powerful API and simple tools, it lets Linux users easily create and manage system or application containers.[2]
The most common app containerization technology is Docker.

1.2 Docker in Go

Docker is a runtime for Linux containers written in Go. Docker uses Linux containers to package application dependencies thus multiple containers can share a single kernel results to decreases of overhead of running applications. Docker chooses Go as its implementation language because Go is a statically typed and compiled language that feels like dynamically typed, interpreted language. Go allows software created in it removing the need for most external dependencies while what Docker want is exactly the same, making deployment easy across various machines.

Although static compilation usually increases the sizes of compiled applications, it makes code more portable since dependencies are included within executables which benefits Docker a lot. The low level interfaces of Go helpers a lot in low level programming like Docker.
However, Go lacks compile-time generics so that will leads to repetition or excessive dynamic code. It also lacks the ability to pin libraries to a specific version. And in Go, maps are not thread safe although they are really fast. The error handling is in Go is also not well developed.

2 Java

Java is one of the most popular, object-oriented programming languages.

Advantages of Java

Java is similar to Go in ways regarding garbage collection and concurrency framework. It can caught errors early and it has a lot of powerful IDEs. These help programmers to easily debug which developing for DockAlt.
Java concurrency is different in that its mechanisms from Go. Synchronization tools for Java is dependent on libraries.
The most important advantage is that Java's static typing can improve the detection of errors,

which can be beneficial for a system that relies on robust code. The code is compiled to bytecode for the Java Virtual Machine allows for code to be platform-independent, which is useful for creating a system like DockAlt.
Java’s popularity, platform-independence, compiled nature, and the reliability are advantages that should be considered for the DockAlt platform.

Disadvantages of Java
Java has no default binding to Linux Containers. It is a difficult for DockAlt to diverse itself from just the Linux Kernel, and still interface well with the extremely low level interfaces of any operating system which might be hard without incorporating different external libraries.
Java’s heavy reliance on the Java Virtual Machine forces systems to include the JVM if they wish to run developed applications. This can lead to heavier-weight systems.
Java does not have duck typing feature, a loss for the ease of programming.

3 OCaml

OCaml is an industrial strength programming language supporting functional, imperative and object-oriented styles.

Advantages of OCaml
OCaml strength is safety, it verifies at compile time that your program is safe, correct apart algorithmic errors. Once it compiles, you are almost sure it is going to work. Go does not give any such guarantee.
And among "safe" languages, OCaml is probably the most efficient, generating fast code running with a small memory footprint.
Moreover, although its development was a little slow in the last years, they have recently restarted working on it, on optimizations and soon multi core support, so I would wait a little before making more comparisons.
OCaml can be compiled to native platform specific code and it can also be compiled to bytecode with a compiler. Compiling to machine-independent bytecode is an advantage for a system like DockAlt since applications can be ported to various systems easily.
It has automatic garbage collection and static type checking reduce the number of errors that are introduced into applications.
These advantages make the DockAlt system reliable and robust.

Disadvantages of OCaml
OCaml does not have a binding for Linux containers. If we want to use OCaml, we have to use an third party library to manage linux container.
OCaml bytecode is usually slower than compiled platform-specific code since the compiler can not make specific assumptions that would improve execution speed on any specific platform.
functional language, get used to the functional paradigm of thinking. relies heavily on its functional nature, would act as a slow start in the development of DockAlt.
Most mutable OCaml data structures do not have well-defined semantics when accessed concurrently by multiple threads. Even data structures that seem like they should be safe but are mutable under the covers, like lazy values, can have undefined behavior when accessed from multiple threads.

4 Scala

Scala is a general-purpose programming language providing support for functional programming and a strong static type system. Designed to be concise, many of Scala's design decisions aimed to address criticisms of Java. [3]

Advantages of Scala
It is object oriented, every value is an object. It is also functional programming paradigms. Scala behaves like most functional languages, and meanwhile allows gradual transition of paradigm.
runs on JVM, Java and Scala classes can be easily mixed, and Java-based tools can work with Scala as well.

Scala is strongly and statically typed. It is thus safe and reliable since its statically typed system allows error checking at compile time, which is preferred option over runtime type checking for large codebases

Disadvantage:
It has no native LXC API binding, we have to build third party library if we want to use Scala.
Scala incorporates features and concepts that are not familiar to many programmers, such as functional programming and continuations. It can take some time to learn these concepts.
There are far fewer developers who already know Scala than who already know Java, which means it will be more difficult if you need to staff up quickly and don't want to spend any time training.

Conclusion

I would like to recommend Scalar to build our DockAlt. Reasons are following:

1 Higher Productivity
Scala programs are from 1/2 to 1/10 the number of lines of code as compared to a functionally equivalent Java program. The larger the application, the more apparent this difference becomes.
2 Higher Quality
Scala encourages a functional programming style, which in turn leads to code with fewer bugs. Also, fewer lines of code means fewer bugs. Fewer bugs means higher productivity and a higher quality product.

3 Asyncio
Scala Actor library and its encouragement of immutable variables help to avoid race conditions among multiple threads and thus improve reliability in concurrent applications.

There is an impedance mismatch between message-passing concurrency and virtual machines, such as the JVM. VMs usually map their threads to heavyweight OS processes. Without a lightweight process abstraction, users are often forced to write parts of concurrent applications in an event-driven style which obscures control flow, and increases the burden on the programmer.
Scala developed event-based actors to unify thread-based and event-based programming under a single actor abstraction. Using advanced abstraction mechanisms of the Scala programming language, we implement our approach on unmodified JVMs. Our programming model integrates well with the threading model of the underlying VM.
Event-based actors support the same operations as thread-based actors, except that the receive operation cannot return normally to the thread that invoked it. Instead the entire continuation of such an actor has to be a part of the receive operation. This makes it possible to model a suspended actor by a continuation closure, which is usually much cheaper than suspending a thread.[4]

Reference:

[1]http://searchitoperations.techtarget.com/definition/application-containerization-app-containerization
[2] https://linuxcontainers.org/lxc/
[3]
https://en.wikipedia.org/wiki/Scala_(programming_language)
[4]http://www.sciencedirect.com/science/article/pii/S0304397508006695