CS 131 Containerization Support Languages Homework 6 Fall 2016

Jahan Kuruvilla Cherian - *University of California, Los Angeles*

Abstract

Docker is a huge open source software containerization platform, originally built on top of the Linux Containers (LXC) to serve the purpose of creating standardized applications by running them in multiple environments within the same host machine, in a fast and lightweight manner. It is new and thus can raise questions to its validity. Thus the aim is to explore alternatives by looking at the languages such as Java, OCaml and Dart, in order to compare it to Go – Docker's original language - in order to build DockAlt.

Introduction

Containerization is the ability to packages up depenpencies, namespaces, cgroups[1] and other application specific libraries into a single container as an image that can then be moved around. This allows for development and deployment onto various types of containers, for testing and debugging, allowing for quick application development. This is what the Linux Containers sought to do for the Linux Kernel, and this is how Docker was born – on top of the LXC through bindings, now with the extra abstraction of the libcontainer. Essentially Docker allows for one host running a specific operating system to run the application within multiple containers that are fast and lightweight in comparison to full Virtual Machines.

In this report we analyze the choice to build Docker in Go, and whether we could substitute this language with another from the set of Java, OCaml and Dart to build our own version of Docker known as DockAlt.

1. Go

Go is a relatively new language developed and released by Google in 2009, with its strengths lying in the support for concurrency through goroutines and channels. The language is neutral [2] in the sense that due to its young age, it has not formed sects within the overall community that strictly argue for it as with more common languages such as Java and OCaml. Go really shines with some of its deeper features such as static compilation, extensive and powerful standard library and heavy duck typing.

1.1 Advantages

Because Go is statically compiled, this means that it compiles down to machine code before being executed. This becomes a huge advantage when it means it requires no additional installations for it to run, and can use popular dynamic libraries that are now on almost every system – such as libc. This makes it easier to test and adopt, and because Docker has the need for ease of transition between systems and platforms, Go really works well. Go even becomes platform independent in the sense of its ability to build for multiple platforms. This removes the burden of having extra dependencies and thus makes it a perfect candidate for bootstrapping, as is the nature of Docker containers. As mentioned earlier, because Go is neutral, it increases the likelihood for user adoption of Docker, since there is no bias against it. One of the biggest advantages of Go is that it contains an official LXC binding, giving it great low-level interfaces to manage processes, system calls and more. Even though Docker now uses libcontainers, it is imperative to have good support for low level interfacing. Docker was built as an open source project and thus it is necessary for it to be readable, maintainable and quick and easy to develop for. Because of Go's strong duck typing, which essentially allows for dynamic type checking to act as type inference, makes development for Docker easy. Go makes the entire development process friendly as upon its installation provides a full development environment by allowing for easy access to the documentation, dependencies on remote servers, tests and more. As mentioned earlier, Go becomes platform independent by providing multiple builds for the specific platforms all from the get-go, without the need for pre-processors. Go provides a very powerful concurrency paradigm through the use of goroutines – modularized functions that are automatically distributed amongst threads – and channels – piping of concurrent goroutines to make the code thread safe.

1.2 Disadvantages

While Go has its advantages, that seem to be extremely beneficial for the purposes of Docker, there remain some drawbacks. While goroutines and channels make concurrency extremely easy and user friendly, they also sometimes sacrifice safety for speed, in which case there would be a compounding effect on the use of channels to make certain data structures such as maps thread-safe. The other key disadvantage is most likely the age of the language itself, as it pales in comparison to the maturity of languages such as Java and OCaml.

2. Java

Java is an age old language focusing its strengths in the object oriented paradigm of working. It has some key advantages such as static typing for better error handling, the plethora of libraries available and more. While Java is a great language, it must be looked at through the perspective of its relevance to creating an application such as DockAlt.

2.1 Advantages

Java has static type checking which could prove to be cumbersome in the writing code stage of the development process. However, the case with most statically typed languages is that it does ease the overall process as it means errors are caught earlier on, in the compilation rather than in the runtime interpreter stage. This means that when developing for DockAlt, programmers will have an easier time debugging code before deployment of the latest build. Java is in some sense extremely portable, in that it gets compiled down to byte code that can then be run on the JVM regardless of the machine architecture at hand. This would mean that multiple build versions would be unnecessary as the JVM would take care of essentially all of the low level machine specific worries. Because of its maturity, Java has a very strong community and a massive library for external modules to be integrated into the application. This could come as both an advantage and a disadvantage to DockAlt development.

2.2 Disadvantages

Because Java is built to be platform agnostic, it has no default binding to Linux Containers. And while this might seem like a benefit, it comes to us as more of a disadvantage. While DockAlt should try and diversify itself from just the Linux Kernel, it still requires the ability to interface well with the extremely low level interfaces of any operating system, which might be difficult without incorporating different external libraries that could bloat the application. Not to mention, the JVM is not on every machine by default. Thus without the JVM coming with the DockAlt system, the Java bytecode would essentially be rendered useless. This means that DockAlt would have to be further bloated to allow for it's Java code to run. The general consensus with statically typed languages is that while they make the program safer, they come at the cost of creating more complex code which in turn increases the development process and could thus lead to a longer time frame needed for DockAlt development.

3. OCaml

OCaml is a multi-faceted language with the ability for object oriented style programming and functional programming. Its strength lies in it's the conciseness and speed of its code.

3.1 Advantages

OCaml is a strong statically typed language [3] with type inference under the hood that is done through compile-time bindings. This means that writing OCaml code becomes extremely fast as you the developer need not type out the types of the data, but just ensure that the operations on the data is consistent. Upon failure of doing so leads to quick error detection during compile time. This seems like the best of both worlds from the Go and Java perspective and might make developing DockAlt quick. Just as is with Java, OCaml is compiled down to bytecode that could be made specific to the system or to an interpreter that comes with the OCaml toolchain. This would be beneficial to an application like DockAlt since it would mean the code is highly portable and machine agnostic. OCaml comes with some extremely powerful libraries that allows it to interface with low-level details – unix library, or sys library.

3.2 Disadvantages

The biggest barrier to using OCaml is to get used to the functional paradigm of thinking. OCaml relies heavily on its functional nature, and for one to truly utilize its capabilities would have to get used to the functional paradigm. This would act as a slow start in the development of DockAlt, as the programmers would have to learn this new paradigm and would have to incur a steeper learning curve as compared to other languages that are more imperative in nature. This could outweigh the benefits of the type inference speedup discussed before. The other key issue is that there is no officially supported library for binding to LXC (there is however a community supported initiative that seeks to create these bindings), which means creating the containerization support – the core of DockAlt – would be a more cumbersome and difficult task than languages with official support such as Go. The final issue is related to the OCaml requiring a Just-In-Time (JIT) compiler or a bytecode interpreter to be associated in the build package for DockAlt, because otherwise it faces similar issues to Java's JVM in that it would require extra dependencies, and make the DockAlt application more bloated.

4. Dart

Dart is a relatively new application programming language that's easy to learn [4], scale and deploy, in order to make common programming tasks easy. It is specifically built as an alternative to JavaScript for web based

applications, and is also in active development by Google.

4.1 Advantages

Dart uses an asynchronous programming paradigm which means that regardless of multiprocessor systems, can make the use of concurrent code extremely effective through event loops, which is something DockAlt might want to do, to be able to launch and run multiple containers simultaneously. It supports optional typing, in the sense it uses dynamic type checking to infer the type based on bindings to built-in types, but can greatly benefit at the programmer's discretion if the type is specified. Being similar to Java and OCaml in the sense that Dart is compiled to bytecode that can be run in the DartVM that comes with Google products, which means it is extremely portable. The fact that it mainly runs on web browsers for web based applications means that if the DartVM comes within the browser (as with Chrome) it could run on any device, which would mean DockAlt would be able to even run on mobile devices at some point. However, this comes with the caveat that exposing low level interfaces might be more challenging. Dart is built to be easy to learn and use which would mean that the developers for DockAlt would be able to work on creating the application much quicker than they would with a language like OCaml, which requires a completely new paradigm of thinking for most.

4.2 Disadvantages

While Dart is working itself up to a strong point, it is still in an infantile stage, and thus may contain several unresolved issues and bugs, as with any new language. There is a lack of LXC support directly for the language. However, there is a lxcdriver and lxc package for the Node.js JavaScript framework, and there exists a dart2js [5] tool to convert Dart to JavaScript, thus allowing for some work around LXC support. This however is a lot more cumbersome and might prove to be slower. As with the other languages, the issue with compiling to bytecode is that the machines that DockAlt would intend to run on would have to have the DartVM installed, or the application itself would have to contain it, thus bloating it up.

Conclusion

Most of the alternative languages for creating DockAlt suffer from somewhat of the same disadvantages, in that they all get compiled to bytecode which could make portability easier, but would incur heavier costs in the application itself. Dart seems to be the easiest language for developers to pick up in comparison to Java and OCaml. All languages provide strong concurrent programming paradigms with Java and OCaml focusing on multithreading and Dart with an asyn/wait mechanism. Dart seems to be a likely contender in the use for developing DockAlt, but faces the key issue of being very new and thus very unpredictable in its nature. Thus my final recommendation would probably be to assess the pros and cons of each language and find what might be worth the sacrifice. OCaml will prove to be great if the developers are willing to put in the time and effort to pass the initial steep learning curve as future code will be concise and efficient; Java will prove to be tried and tested, but might prove to be more complicated to work with due to its verbose nature and Dart would be a step in the right direction for the future of technology, as it shifts its sight to the web, but will suffer from lack of support and libraries necessary for proprietary development.

References

[1] Chenxi, W (2016, May) *Containers 101*. Retrieved from http://www.infoworld.com/article/3072929/linux/containers-101-linux-containers-and-docker-explained.html

[2] Jerome, P (2013, November) *Docker and Go. Retrieved* from http://www.slideshare.net/jpetazzo/docker-and-go-why-did-we-decide-to-write-docker-in-go/27-maps_arent_threadsafe_deliberate_decision

[3] "Strong Static Typing with Type Inference" *OCaml for the Skeptical*. Retrieved from http://www2.lib.uchicago.edu/keith/ocaml-class/static.html

[4] "Core goals" *Dart*. Retrieved from https://www.dartlang.org/

[5] "The Dart-to-JavaScript Compiler" *Dart WebDev*. Retrieved from https://webdev.dartlang.org/tools/dart2js