A New Language for DockAlt

Jun Zhou 804179311

Intro:

Linux Containers are a sort of compromise between creating a chroot and a VM for an application. They can create environments for each app close to a standard Linux installation, but are more lightweight than VMs because each container does not require a separate kernel.

Docker is one container implementation. It wraps software in an environment with its own bin/libs, all on top of the Docker engine, which runs on top of the host OS. The goal is to guarantee that software will always run the same, regardless of the actual machine it is ported to. However, Docker containers still share the OS kernel, filesystem and other resources, so containers are lightweight. Docker containers can run on all major Linux distributions, with the added benefit of extra security. It is more lightweight than a VM which involves each app having its own guest OS and kernel on top of the host.

In this report, we analyze three other possible languages that would be feasible for writing DockAlt, an intended backup or alternative implementation of Docket. The original Docker is written in Go, and any alternative language choice should try to address the considerations which caused the writers to choose Go in the first place. Specifically, the languages under consideration are Java, Ocaml, and Dart.

Why Go:

These are the main reasons the developers of Docker chose Go.

1) Go compiles statically to create a static binary. This means that generally, builds will run without needing to install other dependencies on a new machine. The exceptions are dynamic libraries like cgo.
2) Apparently, Go is a neutral language, unlike C++, Python, Ruby, and Java. I'm not 100% sure what this means. I think programmers tend to have strong opinions about the more popular languages, while Go is just a safer choice and easy to pick up.
3) Go has asynchronous primitives and low-level interfaces. This includes functionality like waiting for I/O, handling processes, syscalls, etc.
4) Go uses duck typing, allowing run-time type checking. An advantage of duck typing is concise code and possibly flexibility. A drawback is that dynamic type checking is slower.
5) Go allows a multi-architecture build and does not require a preprocessor. This cuts out a step in the compilation process and avoids file size increases which can happen at the preprocessing step.

Possible Alternatives:

Java is a portable language that provides some of the benefits of Go. Although code can compile into object files rather than always compiling into static binaries like Go, programs can still run on any machine with a JVM. Java is also a well-defined language, avoiding some of the semantic ambiguities of languages like C/C++. On the downside, while Go is considered somewhat easy to learn, Java is more complex. Java is also strongly and statically typed. While the Docker developers cited duck typing as an advantage of Go, static typing has its own advantages, including safety. Java also includes a lot of libraries that can provide the functionality needed to write a Linux Container. It can handle low level functionality including I/O, processes, and system calls. Like Go, Java also does not have a preprocessor. Because Java runs on the JVM, it may be slower than a Go binary. It is also more verbose than Go.

Ocaml, like Go, compiles into bytecode. This makes it portable and fast, giving it an advantage over Java. Like Java and unlike Go, Ocaml is statically type checked. This makes code very safe, but may slow down development time, because a single error makes the code un-runnable, even for testing portions that do not run into the error causing

branch. Ocaml also includes libraries supporting system calls and other low level functionality. Like Go, Ocaml is also very concise, but perhaps at the cost of readability. Java is more verbose, but possibly more readable. Given its similarities to Go and Java, but also its additional advantage of speed, conciseness, and type safety, I would pick Ocaml over both Go and Java.

Dart is a language I have not used before, but at first glance, the syntax looks simple and familiar, almost like C++ or Java. It provides optional static type checking, based on whether a type keyword is placed in front of a variable or omitted, striking a good balance between safety and flexibility. It is object oriented, like Ocaml and Java. Dart also includes a process class for starting new processes and includes libraries for I/O and asynchronous programming. However, it seems that Dart has previously been more commonly used for web programming, and can optionally compile to Javascript. Like Java, Dart has a DartVM. Recently, Google has been using it to build client and server apps and can be compared to Node.js. One possible drawback of Dart is that it is a fairly new language. Writing DockAlt in Dart would not necessarily be a safer bet than using Docker in Go.

Conclusion:

I would probably recommend either Ocaml or Java for DockAlt. Dart's optional static type checking seems very useful, but for reliability, perhaps an older language is better. After all, we are only writing a backup to Docker. Between Ocaml and Java, I would probably choose Ocaml, which is more similar to Go in its compilation to a static binary. Ocaml programs can then run faster than Java. Ocaml also has the benefit of being a very concise language with strong static type checking.

References:

[1] *Docker and Go: why did we decide to write Docker in Go?* http://www.slideshare.net/jpetazzo/docker-and-go-why-did-we-decide-to-write-docker-in-go

[2] *Java's three types of portability*, http://www.javaworld.com/article/2076944/java-s-three-types-of-portability.html

[3] *Java programs vs Go*, https://benchmarksgame.alioth.debian.org/u64q/compare.php?lang=java&lang2=go

[4] *Ocaml for the Skeptical*, http://www2.lib.uchicago.edu/keith/ocaml-class/why.html

[5] *A Tour of the Dart Language*, https://www.dartlang.org/guides/language/language-tour

[6] *Google Dart Will Not Be the Next Default Programming Language of the Web*, https://arc.applause.com/2015/03/27/google-dart-virtual-machine-chrome/

[7] *Linux LXC*, https://linuxcontainers.org/lxc/