Language Options for DockAlt, a Containerization Alternative to Docker

Harrison Liddiard
University of California, Los Angeles
Winter 2017

Abstract

This executive summary analyzes the viability of three programming languages as candidates for DockAlt, an in-house implementation of Docker: Java, OCaml, and Scala. Go, the language in which Docker itself is implemented, is also examined.

1. Introduction

Docker is an open-source container management platform. Like virtual machines, containers abstract the applications running in them from the physical hardware for easier dependency and environment management. Unlike VMs, though, containers run on the host operating system kernel rather than using their own which reduces overhead. [1]

Docker aims to solve the problem of managing multiple components of an application (e.g., database server, cache, application server, front-end application), each of which must be run across multiple environments (e.g. developers' local machines, QA servers, production). This arrangement leads to an NxM "matrix" where N = the number of discrete application components and M = the number of environments in which it runs. [2]

Leveraging Linux containers (LXC), Docker uses images that run processes in isolation for each component of the application. [3] Docker maintains an image registry from which developers can pull common images (such as Apache, PostgreSQL, etc.) Developers can also publish to this public registry with a single "docker push" command.

2. Docker's Implementation (Go)

Docker is written in Go, an open source programming language created by Google and released initially in 2009. [4] The language features static typing, garbage collection, and memory safety. It is a compiled language. [5]

2.1 Advantages

In a tech talk, the Docker developers explained their reasons for choosing Go and some of the shortcomings they encountered. Among the primary advantages they cite are static compilation, low-level interfaces, a robust standard library, and utilities that help with the entire development workflow, from prototyping to testing to documentation. [2]

Go's static compilation makes it easy to install on a variety of operating systems – it compiles to a binary so no it is not dependent on the presence of an interpreter or a VM to run.

Go has many tools for asynchronous operations and concurrency. It has its own scheduler which is optimized specifically for Go code to be more performant than the OS scheduler. [6] "Goroutines" are functions that can run concurrently with less overhead than threads that make use of Go's scheduler. Functions running as goroutines can send information to each other using channels. [7]

Go uses "duck typing," which enables the developer to create a custom type that implements a specific interface. Simply by providing the interface methods expected, this new type can be used anywhere that expects an object with that interface. [8]

Finally, its package ecosystem, while less developed than more mature languages like Java and Python, is robust. The language comes with package management built in via the "go get" command which can download and install packages from GitHub and other sources. Package documentation is also built in – the "go doc" command fetches the documentation for any Go package. [2]

2.2 Disadvantages

Go's package manager lacks the ability to pin libraries to a specific version which impedes reproducible, deterministic builds. It also does not handle private repositories. Go does not have a way to run cleanups in tests, necessitating incomplete workarounds. Unlike other languages like Java, Go has no IDEs. [2]

3. Java

3.1 Advantages

Java is one of the most widely used programming languages in existence. [9] Developers who are fluent in Java would be easier to find than those who are similarly experienced with Go.

Java features a strong, static type system which makes it straightforward to catch type-related errors before runtime. [10] Java has many well-developed IDEs including Eclipse, IntelliJ IDEA, and NetBeans. [11] Packaging applications is fairly simple with the widely supported, self-contained Java Archive (JAR) file format. [12] The Java Virtual Machine (JVM) enables applications to work cross-platform by abstracting OS-specific system calls away from the application code. [13]

3.2 Disadvantages

Java's largest disadvantage for this specific project is its lack of support for LXC. While third-party implementations exist, they rely on executing LXC commands in a shell and parsing the text output back in Java. [14] This is not nearly as efficient as interfacing directly with LXC's C API and could be a significant drawback for an application that relies heavily on functionality from LXC. For adequate performance, this functionality would likely have to be written from scratch using Java Native Interface (JNI), which provides interface with C code. [15] JNI, however, has its own issues including difficulties in debugging and error recovery. [16]

Additionally, Java is overall the most verbose language of the options discussed. While there are benefits to the explicit nature of Java's syntax, many argue that it goes too far. [15]

Dependency management is not quite as straightforward with Java, either. It is not built in to the language as with Go, and does not have a simple, lightweight package manager like Python (pip) or JavaScript (npm). Maven is the most widely used package manager, but it is more of a massive project management tool. [16] While this full-featured software has its uses, the learning curve is high. Whereas project requirements for package managers like pip and npm be specified in a short plaintext or JSON requirements file, Maven projects are configured with an XML file called a Project Object Model ("POM") which starts off with 110+ lines of configuration and can quickly become even larger. [17]

4. OCaml

4.1 Advantages

OCaml is a primarily functional language with support for object-oriented design along with imperative-style code. [18] OCaml uses strong static typing with type inference, which can be a major advantage in catching type-related errors at compile time, leading to more reliable code without requiring type notation as verbose and prevalent as Java's. [19]

Another important advantage that OCaml has for this particular software project is its foreign function interface. This feature of OCaml enables OCaml code to interface directly with C libraries, mixing OCaml's higher-level code with lower-level C bindings. [20] This would allow a fairly straightforward interface with LXC.

4.2 Disadvantages

Despite its easy interface with C, OCaml does not appear to have an LXC binding library that is actively developed. A cursory search shows a few open source projects, but they do not appear

to have much activity and lack tests. [21] This means that while these project would likely be useful as a reference, but it would be unwise to rely on them as a foundation for DockAlt and developer time would need to be invested to either build upon an existing project or implement this functionality from scratch.

Additionally, OCaml is simply not as popular as the other languages compared in this report. By the TIOBE index, OCaml does not make the top 50. [9] This is reflected in its lack of community and libraries compared to the expansive Java ecosystem. OCaml's community is still substantial, but its relatively low popularity would make finding skilled OCaml developers and actively maintained libraries more difficult.
-memory allocation/GC not explicit

5. Scala

4.1 Advantages

Scala is a functional and object-oriented programming language that runs on the JVM. [22] This means it enjoys all the benefits of Java code's inter-platform operability. Like Java, Scala has strong static typing, meaning more errors can be caught at compile time for more reliable code.

Developers who are familiar with functional programming will have no problem getting on board with Scala. Syntactically, it shares similarities with other functional languages like OCaml and Haskell, and it features type inference, function currying, and immutable data. [23] Unlike Java, Scala's functional nature means code is generally free from side effects, making it easier to reason about and debug. Scala was also created with a conscious effort to avoid some of Java's shortcomings and edge cases. [24]

4.2 Disadvantages

Unfortunately for this particular project, Scala suffers the same major disadvantage as Java – it has no native LXC API binding. This would make it difficult to implement an application that relies heavily on interfacing with LXC.

Scala's functional nature, while advantageous to developers who have worked with and enjoy functional programming, could also present a steep learning curve to those who haven't. With a completely different programming paradigm to the predominantly imperative nature of Java, Python, and Go, training would be necessary that could push back development on DockAlt.

6. Conclusion

Of the three languages examined plus Go, OCaml and Go seem to be the best candidates for implementing DockAlt. Despite its relatively small community, OCaml's straightforward means of interfacing with C code are compelling for a project that will rely heavily on interfacing with LXC which has a C-based API. Its functional nature and type inference are also considerations in its favor.

Go, as explained by the developers of Docker, is also a language worth considering for this application. A young but promising language, Go provides a good low level interface while having better inherent memory safety than languages like C. However, Go's lack of maturity is apparent in its lack of key features for testing and packaging, missing tools that languages like Java and Scala have had for years.

In addition to the languages examined here, it may also be worthwhile to look into Haskell, Python, and Rust as candidates. Haskell is very syntactically similar to OCaml, but it has a more robust community and an LXC interface library that appears more actively maintained. [27] Python is very different from the languages examined here, but it is widely used and has an actively maintained LXC binding. [28] Rust, like C, is a lower-level programming language, but unlike C, it guarantees memory safety, and it has a robust foreign function interface. [29]

References

[1] "What Is Docker?" *Docker*, Docker Inc., 29 Nov. 2016, www.docker.com/what-docker.

[2] Petazzoni, Jérôme. "Docker and Go: Why Did We Decide to Write Docker in Go?" *SlideShare*, LinkedIn Corporation, 7 Nov. 2013, www.slideshare.net/jpetazzo/docker-and-go-why-did-we-decide-to-write-docker-in-go/.

[3] Avram, Abel. “Docker: Automated and Consistent Software Deployments.” *InfoQ*, C4Media, Inc., 27 Mar. 2013, www.infoq.com/news/2013/03/Docker.
[4] Kincaid, Jason. “Google's Go: A New Programming Language That's Python Meets C++.” *TechCrunch*, AOL, 10 Nov. 2009, techcrunch.com/2009/11/10/google-go-language/.
[5] “Frequently Asked Questions (FAQ).” *The Go Programming Language*, Google, golang.org/doc/faq.
[6] Morsing, Daniel. “The Go Scheduler.” *Morsing's Blog*, Daniel Morsing, 30 June 2013, morsmachine.dk/go-scheduler.
[7] Doxsey, Caleb. “Concurrency.” *Go Resources*, Caleb Doxsey, www.golang-book.com/books/intro/10.
[8] Ryer, Mat. “Duck Typing in Go.” *Medium*, Medium, 3 June 2016, medium.com/@matryer/golang-advent-calendar-day-one-duck-typing-a513aaed544d.
[9] “TIOBE Index for December 2016.” *TIOBE - The Software Quality Company*, TIOBE Software, 2016, www.tiobe.com/tiobe-index/.
[10] “Category:Statically Typed Programming Languages.” *Wikipedia*, Wikimedia Foundation, en.wikipedia.org/wiki/Category:Statically_typed_programming_languages.
[11] “Java IDEs.” *Wikibooks*, Wikimedia Foundation, en.wikibooks.org/wiki/Java_Programming/Java_IDEs.
[12] “Using JAR Files: The Basics.” *Java Documentation*, Oracle, 2015, docs.oracle.com/javase/tutorial/deployment/jar/basicsindex.html.
[13] “The Java® Virtual Machine Specification.” *Java SE*, Oracle, 28 Feb. 2013, docs.oracle.com/javase/specs/jvms/se7/html/index.html.
[14] Hamshawi, Waseem. “Waseemh/Lxc-Java.” *GitHub*, GitHub, Inc., 8 Feb. 2016, github.com/waseemh/lxc-java.
[15] “Java Native Interface Overview.” *Java SE Documentation*, Oracle, 12 Nov. 2002, docs.oracle.com/javase/7/docs/technotes/guides/jni/spec/intro.html#wp725.
[16] Glen. “Disadvantages of Using Java Native Interface.” *JNI - Disadvantages of Using Java Native Interface*, Stack Exchange, Inc., 8 Sept. 2009, stackoverflow.com/a/1393953/2487925.
[17] “Why Is Java so Verbose?” *Quora*, Quora, www.quora.com/Why-is-Java-so-verbose.
[18] Porter, Brett et al. “Welcome to Apache Maven.” *Maven*, Apache Software Foundation, 2016, maven.apache.org/.
[19] “Introduction to the POM.” *Maven*, Apache Software Foundation, 2016, maven.apache.org/guides/introduction/introduction-to-the-pom.html.
[20] Agarwal, Ashish, et al. “OCaml Is an Industrial Strength Programming Language Supporting Functional, Imperative and Object-Oriented Styles.” *OCaml*, 3 Aug. 2016, ocaml.org/.
[21] Waclena, Keith. “Type Inference and Type Errors.” *OCaml for the Skeptical: Type Inference and Type Errors*, University of Chicago, 17 June 2006, www2.lib.uchicago.edu/keith/ocaml-class/errors.html.
[22] Minsky, Yaron, et al. “Foreign Function Interface.” *Real World OCaml*, O'Reilly Media, Nov. 2013, realworldocaml.org/v1/en/html/foreign-function-interface.html.
[23] Zoggy. “Zoggy/Ocaml-Lxc.” *GitHub*, GitHub, Inc., 25 Sept. 2015, github.com/zoggy/ocaml-lxc. Accessed 11 Mar. 2017.
[24] “Scala.” *The Scala Programming Language*, École Polytechnique Fédérale De Lausanne (EPFL), www.scala-lang.org/.
[25] “Introduction.” *Scala Documentation*, EPFL, 2015, docs.scala-lang.org/tutorials/tour/tour-of-scala.html.
[26] Martin Odersky et al., *An Overview of the Scala Programming Language, 2nd Edition*.
[27] “The LXC Package.” *LXC: High Level Haskell Bindings to LXC (Linux Containers).*, 21 Jan. 2016, hackage.haskell.org/package/lxc.
[28] “Python-LXC.” *GitHub*, GitHub, Inc., 4 Jan. 2017, github.com/lxc/lxc/tree/master/src/python-lxc.
[29] “The Rust Programming Language.” *The Rust Programming Language*, 9 Feb. 2017, www.rust-lang.org/en-US/.