CS 131 Homework 6: Containerization support languages

Zhehao Wang, 404380075
zhehao@cs.ucla.edu

Abstract

In this report, we inspect alternative languages to implement Docker, an open platform for building, shipping and running applications "anywhere". The original Docker is implemented in Go, with features including utilizing Linux containers to build lightweight program execution environments, avoiding the dependency hell problem, and easy to use, develop and maintain [1]. This work studies the feasibility of using Java, Python or Scala to achieve the same functions as Docker does, and analyzes the pros and cons of each choice.

1 Docker in Go

Before we dive into details of the alternative languages, it's important to understand what Docker hopes to provide, why it's significant, and why the developers of Docker chose Go language.

As the scale and complexity of software grows, software management and deployment have become increasingly difficult. Commercial softwares today may face a variety of platforms, each with its own set of libraries providing certain functions to applications. For developers with the goal of distributing their software to different platforms, specific research usually need to be done for that platform, and often times different binaries will be needed for different platforms. To make matters worse, sometimes dependency issue's out of the control of the application developer: for example, when a dependency's not available or buggy for a platform.

The usual answer for this problem involves using a virtual machine. We can distribute a fully functioning virtual disk that carries the application as well as its dependencies, and the environment it's running under. The idea of Docker is similar, only that it's achieving this purpose with a more lightweight approach using Linux containers [3], an OS level virtualization environment for running multiple isolated Linux systems on a single host. A Linux container provides its own process space and network interface for Docker applications. [2]

Before choosing Go, the developers inspected other choices like Python, Ruby, Java, etc. Python is used in the developer's previous project, dotCloud, but the developers wanted to start from a clean slate with a more neutral, and less controversial language. An important concern here is adoption by ops: having a single binary that you can drop in any server is a big win, but Ruby shops don't use Java, Python shops don't use Node, etc [6]. Go is neutral in the sense that it's mostly platform agnostic, making it more suitable for this case. Go is designed with static typing, "be light on the page" and "support networking and multiprocessing natively" in mind, and features [5]

- A C-like syntax, but more concise and readable
- An expressive and lightweight type system that uses duck typing, and removes type hierarchy
- A hybrid stop-the-world/concurrent mark-and-sweep garbage collector
- A toolchain that by default generates statically linked native binaries without external dependencies
- Built-in concurrency primitives: goroutines and channels
- Remote package management system and hierarchical package naming
- Fast compilation because of lightweight type analysis, and easy dependency analysis [4]

Among these features, the developers of docker emphasized that a good package management system, a good standard library, and easy to read and contribute among their top reasons for choosing Go. [6]

2 DockAlt in Java

Java is one of the most widely used programming languages that has the following features.

- Java is imperative and object oriented. [7]
- Compiler compiles Java source code into byte code, which gets executed in a Java Virtual Machine.
- Java is strongly and statically typed. It has type hierarchy, differentiates primitive and reference types, and is enhanced with explicit exception marking. [8]
- Garbage collector is up to JVM implementation. Some commonly used JVMs have concurrent mark and sweep garbage collection.
- Native library has a large number of built-in classes that can handle various requirements.

The pros and cons of implementing DockAlt in Java include:

- Pro: Implementation safety and reliability. In a large codebase like DockAlt's, this is an important concern. Java type system helps in improving safety. By having a strongly typed language and compile time type checking, mistakes are more easily detected and reported at compile time. Java also has explicit exceptions in the type, which requires that exceptions thrown by the callee should be handled or passed on in the caller function, this is another feature that would improve code reliability. However, it's worth mentioning that because of the regulations this type system introduced, compilation tends to be slower. (Go compiler is fast, with one reason being that type relationships, such as the type hierarchy, is removed)
- Pro: JVM provides some platform agnosticity. Java code can run on various platforms, mainly because of the existence of a corresponding JVM for that platform. This means for all the platforms that can run a standard JVM, only one version of DockAlt source is needed. Platform agnosticity is a plus for DockAlt, and in this sense Java is similar with Go.
- Con: Rather verbose code, longer development cycle and not very easy to use. Java may not be a good option for rapid prototyping purposes, or attracting community contributions. Docker as of right now has a large community, whose growth could be hindered if Java is chosen instead.
- Con: Lacks LXC API. Although Java has a good number of useful built-in libraries, LXC interface in Java come as third party libraries, or bindings wrapping around an implementation in a different language. This will slow down the development cycle, add to the list of things that could go wrong, and add another factor that's potentially out of the control of the DockAlt developers.

3 DockAlt in Python

Python started out as a scripting language, and is often used for fast prototyping [9]. The features of Python include:

- Python is imperative and object oriented.
- Python interpreter interprets and executes Python code at runtime. (To be specific, though, a Python byte code .pyc could be "compiled" from Python source code, and this pyc can be interpreted (CPython implementation), or just-in-time compiled. Python source can also be compiled to byte code for different platforms, like IronPython for .Net.).
- Python type system uses duck typing, and is strongly, dynamically typed. [10]
- Standard Python implementation uses reference counting for garbage collection, with a supplemental GC facility to deal with reference cycles.
- Python syntax is designed to be highly readable, with a relatively uncluttered layout. It uses English keywords frequently, and indentation to delimit program blocks. [11]
- Python comes with good remote package management. (de-facto management program, pip, now ships with the official Python distribution)

The pros and cons of implementing DockAlt in Python include:

- Pro: Good remote package management system. As of now "Go get" cannot specify a certain version of a package, and the developers of Docker mentioned this as a painful issue to deal with, before they set up repositories with their custom versions of dependencoes. Python pip offers more customizability, as well as a large set of packages for different purposes. These packages include LXC interfaces, which could be used for fast prototyping of DockAlt.
- Pro: Easy to use, easy to read and contribute. Python is usually advertised for its simplicity, since syntax design of Python placed heavy emphasis on

simplicity. This is a major reason that contributed to Python's success, and it'll likely make prototyping DockAlt easier, as well as attract a larger community, and making contributions from them easier to incorporate as well.

- Con: Python interpreter could have rather bad performance. Interpretation is relatively slow, because the source code is not pre-compiled into a binary, but both analyzed and executed at runtime. Although many applications would sacrifice performance for code simplicity and readability, DockAlt may not be one of them, as when providing a lightweight platform on which various applications may be tried out, it's usually a plus if the platform implementation itself is as efficient as possible.
- Con: Less reliable. Because Python is dynamically typed, type errors are caught at runtime, which may escape notice if the bugged code happens to be not covered in certain test runs. This is generally an unwanted feature for large software projects. However, this can be worked around with well designed software testing frameworks and test suites.

4 DockAlt in Scala

Scala advertises itself as a language with the advantages from object oriented and functional programming paradigms [12]. Its features include

- Scala is objective oriented, every value is an object, and every operation is a method call.
- Scala is functional. It has first-class functions, which means function are objects in Scala, and function type is just a regular class. Like most functional languages, Scala can do pattern matching over arbitrary classes, and has a general preference of immutability over mutation. Meanwhile, Scala allows gradual transition of paradigm: an imperative-language-like programming style is allowed.
- Scala runs on JVM, Java and Scala classes can be easily mixed, and Java-based tools like ant, Eclipse, etc work seamlessly with Scala as well.
- Scala is strongly and statically typed, and its type system implements parametric polymorphism and type inference. [13]

Given these features, we summarize the pros and cons of using Scala as

- Pro: Platform agnosticity. Since Scala byte code runs in JVM, it inherits the platform agnosticity advantage introduced by JVM.
- Pro: Safety and reliability. Similar with Java, a strongly and statically typed system allows error checking at compile time, which is the preferred option over runtime typechecking for larger codebases.
- Pro: Inter-operability with Java. Java has a variety of useful built-in and third party packages. Since Scala can import from and export to Java classes, it would offer a wider range of tools to choose from, a rather mature native library to handle common needs, and the compiled package from Scala would be usable for the Java community as well.
- Con: Not so easy to use. Even though Scala allows an imperative programming style, its core is still functional; and functional programming paradigm should be applied to utilize the language's efficiency. These facts make the language harder to users familiar only with imperative languages. And given that the language has a smaller community than popular imperative languages like Python or Java, it's likely to attract less community contribution as well.

References

[1] Docker website, https://www.docker.com/.

[2] Introduction to Linux containers, https://linuxcontainers.org/.

[3] Docker on Github, https://github.com/docker/docker.

[4] Go website, https://golang.org/.

[5] Go in Wikipedia, https://en.wikipedia.org/wiki/Go_(programming_language).

[6] GopherCon 2014 Making Docker GO: Why One of the Fastest Growing Open Source Projects, https://www.youtube.com/watch?v=i26SYvVu1nw.

[7] Java in Wikipedia, https://en.wikipedia.org/wiki/Java_(programming_language).

[8] Course material: introduction to Java type system, http://www.ccs.neu.edu/home/riccardo/courses/csu370-fa07/lect4.pdf.

[9] Python in Wikipedia, https://en.wikipedia.org/wiki/Python_(programming_language).

[10] Why is Python a dynamic language and also a strongly typed language, https://wiki.python.org/moin/Why%20is%20Python%20a%20dynamic%20language%20and%20also%20a%20strongly%20typed%20language.

[11] The Zen of Python, https://www.python.org/dev/peps/pep-0020/.

[12] An introduction of Scala, http://www.scala-lang.org/what-is-scala.html.

[13] Types in Scala, https://twitter.github.io/scala_school/type-basics.html.