CS 131 Homework 6. Containerization Support Languages

Yuxing Chen
University of California, Los Angeles

Abstract

Containerization is to use container to package everything required to make a piece of software run into isolated containers, which makes systems efficient and lightweight [2]. It also guarantees software will always run the same. Docker is a container platform, which makes use of Linux Container (LXC) [2]. Docker is implemented using the Go programming language. This report is going to discuss the reasons for using Go to implement Docker as well as the drawbacks. Then it will examine three possible alternatives to Go: Java, OCaml, and Scala. We will analyze the technologies' effects on ease of use, flexibility, generality, performance and reliability for these four languages.

1 Introduction: Docker

As mentioned in the abstract, Docker is a container platform built on top of Linux Container (LXC) [2]. LXC creates "an environment as close as possible to a standard Linux installation but without the need for a separate kernel" [6]. Docker uses the resource isolation features of the Linux kernel such as `cgroups` and `kernel namespaces` "to allow independent containers to run within a single Linux instance, avoiding the overhead of starting and maintaining virtual machines" [4]. With Docker, users can run and test their programs without worrying about damaging the host system. Docker containers are lightweight and fast. They take up fewer resources. We can regard Docker as a lightweight virtual machine, which takes its own process space, network interface, etc.

2 Go and Docker

Docker is implemented with the Go programming language. Go is a compiled, statically typed language created at Google.

2.1 Advantages

Go is a *compiled, statically typed* language. It is easy to install, test, and adopt. Since Go is statically typed, errors can be found at compile time, which provides more reliable code. It also improves the efficiency. As a compiled language, code written in Go executes faster. Meanwhile, Go comes with a *garbage collector*. Therefore, users don't need to worry about memory allocation and deallocation, which improves coding efficiency and safety. Dealing with `I/O` or network events is expensive and painful. With *good asynchronous primitives*, Go frees developers from writing their own asynchronous designs to improve efficiency. With *low-level interface*, Go can better communicate with Linux Container (LXC) like managing processes and doing `syscalls`. Go offers *extensive standard library* and data types. Uses can apply these developed functionalities to develop their own programs, which helps users save time in writing code and makes Docker easy to use. Also, due to abundant available libraries, developers may find writing programs in Go has good flexibility. Go is a language that uses *strong duck typing*, which makes type checking easier in Go [3]. Last but not the least, LXC provides APIs for Go [6].

2.2 Disadvantages

While having many advantages, Go has several drawbacks. In general, Go is a relatively new language when compared to mature languages like Java and C. Therefore, its immaturity may lead to some performance issues.

The first problem is thread-safety. As mentioned in the insider-view talk, maps are *not thread-safe* in Go [3]. It's up to developers to ensure that they are safe, either using `sync.Mutex` to protect processes or using channels of channels, which multiples the coding workload and may bring safety issues. Since there is no IDE in Go, developing a program and testing are separated, which hurts

programmers' productivity. Meanwhile, the error handling in Go can be verbose, which increases coding difficulty. What's more, the command go test in Go cannot have destructors or cleanups in test and does not work too well when running individual tests. In Go, it is not easy to build multiple binaries when they share some common code. According to the insider-view talk, "each program has to be in its own package, and package must be 'main'" [3]. However, based on the idea behind Docker, it gives all its dependencies to users, so users need to import source code into their own repository, which increases the code size and sacrifices efficiency.

3 Java and DockerAlt

Java can be a plausible candidate for implementing DockerAlt, an alternative to Docker implemented in Go. Java has been developed for a much longer time than Go. Its maturity brings developers many advantages that Go does not have. However, Java also has some downsides when it is used to implement our DockerAlt.

3.1 Advantages

Like Go, Java is also a *statically typed language*. Therefore, it detects type errors during compile time, which improves code reliability and efficiency as Go does. Besides, Java has much longer history than Go does, and generations of users keep contributing to it. In this sense, Java is a more mature programming language and its compiler can probably capture more errors, which further improves the reliability of our code. As a mature language, Java provides *extensive libraries*. These well-supported libraries ease users from writing massive and complicated code, which also helps improve the efficiency. In addition, there are easy ways in Java to *reuse code*. Thus, Java also obtains great flexibility. Java obtains *great multithreading capability*, which enables users to build applications with many concurrent threads of activity. A better support for multithreading makes the programs fast, which improves performance of our code. In this sense, using Java to implement DockerAlt will give us better performance. Unlike Go, Java has many open source IDEs, which makes testing and debugging much easier than doing the same things in Go. Meanwhile, there are exceptions and error handling in Java, which makes program easier to read, test and debug. Java runs on JVM (Java Virtual Machines) [5]. JVM makes it possible that one program written in Java can run on any other Java enabled system without adjustments. This gives us better and easier multi-platform support, which is a fundamental but important consideration in developing DockerAlt.

3.2 Disadvantages

Since Java runs on the virtual machine, DockerAlt implemented with Java would not be very convenient to communicate with low level applications. However, the Linux container (LXC) that we are going to build DockerAlt on is a low-level software. Meanwhile, according to the documentation of LXC, LXC does not give default API binding to Java [6]. However, Go has API support from LXC. From this aspect, the implementation of DockerAlt using Java might be hard than the implementation of Docker using Go.

4 OCaml and DockerAlt

OCaml is another plausible candidate for implementing DockerAlt. OCaml is very different from the first two languages discussed in this paper: it is a functional programming language.

4.1 Advantages

Like Go and Java, OCaml is *statically typed language*, which improves code efficiency and reliability as mentioned in previous sections. However, unlike Java, OCaml is implicitly typed, which means the types of bindings do not need to be written down. It uses the tool of type inference. This is very convenient and efficient, providing good support for high-order functions [14]. OCaml also provides *garbage collection*, so users do not need to worry about storage management, which makes the code more reliable and frees developers from doing allocations and deallocations. The evaluation order of functions does not matter in OCaml. Also, functions in OCaml do not have side effects. Based on these two features, OCaml code can be *easily parallelized*, which makes our program run faster. OCaml is a programming language that supports "functional, imperative and object-oriented styles" [7]. This is a feature that neither Java nor Go has. Since OCaml supports all three programming styles, it gives more flexibility in dealing with different purposes. The command `ocamlc` in OCaml compiles the OCaml source files into bytecode object files [1]. The executable files linked by `ocamlc` are then run by the byte interpreter `ocamlrun`. The bytecoded system currently "runs on any POSIX-compliant operating system with an ANSI-compliant C compiler" [9]. Besides, the native-code compiler in OCaml supports several processor/operating system combinations. Thus, there are many platforms that are supported by OCaml. This would be good for implementing DockerAlt, since it enables higher portability. There are IDEs available for OCaml. Ocaml has a built-in debugger `ocamldebug` [8].

OCaml also provides exceptions. Therefore, unlike Go, it is not hard to test and debug.

4.2 Disadvantages

Same problem as in Java, Linux container (LXC) does not provide native API for OCaml, which may yield some security and reliability issues if using third party APIs [6]. OCaml is not designed as a low-level language, which means the communication between program written OCaml and low-level applications would be hard. However, our DockerAlt should be built on top of LXC, an operating-system-level virtualization method. Thus, it might be harder for developers to write DockerAlt in OCaml than implement Docker in Go.

5 Scala and DockerAlt

Scala is an object-oriented *and* functional programming language [13]. Scala interops seamlessly with Java and runs on JVM (Java Virtual Machine) [13]. From the documentation, we can tell that Scala is a Java-like language which "unifies objected-oriented and functional programming" [12]. It shares some advantages and disadvantages with Java in implementing our DockerAlt, while having a few features that are different from Java.

5.1 Advantages

Same as Java, Scala is a *statically typed language*, so the errors are found at compile time, which ensures the reliability of our code. Scala is also similar to OCaml in the sense that it uses *type inference*, which is convenient and efficient. Type inference frees programmers from doing type annotations, making coding easier. Type inference also better supports higher-order functions [15]. Scala itself does not have the responsibility of garbage collection. Instead, JVM that Scala runs on gets garbage collected quickly [11]. Therefore, Scala is safe and reliable in memory management. Meanwhile, Scala has *lazy evaluation*. It could be an advantage since lazy evaluation skips useless or dangerous computation. Scala is compatible with current Java programs. As mentioned in the introduction part for Scala, Scala and Java interops seamlessly. It takes the well-supported and extensive Java libraries, which not only makes users easier to develop more features but also maintains great flexibility in coding. In addition, compared to Java, Scala syntax is more concise and extensible, which reduces the code size. Scala further extends the *concurrency* support in the Java language. Many data types in Scala are *immutable by default*, which ensures thread-safety and these data types can be shared easily [10]. Scala also provides abstractions like `ThreadRunner` and `Future` for doing thread-based concurrency [10]. It also has actor-based concurrency model, where `Actors` are lightweight processes [10]. These processes are guaranteed to be scheduled on at most one thread for execution, so there is no need for synchronization. The isolated mutability helps simplify concurrent programming [10]. In this case, DockerAlt implemented in Scala is likely to have better performance. It is easy to test and debug the Scala code, since there are IDEs available for Scala.

5.2 Disadvantages

Same problem as in Java, Linux container (LXC) does not provide official API for Scala, which may yield some security and reliability issues if using third party APIs. Scala is a Java-like language and runs on JVM. Therefore, programs written in Scala would be not very convenient to communicate with low-level applications like LXC, which brings a downside when implementing DockerAlt in Scala as in Java. Therefore, comparing with Docker's implementation in Go, developers may encounter more issues if they use Scala to implement DockerAlt.

6 Conclusion

Based on the discussion above, we can see that each of these four languages has both advantages and downsides. I think Go might still be the best language to implement Docker, since Go has lower-level interfaces and API support from LXC. These are qualities that Java, OCaml and Scala do not have. However, Go is not as mature as languages like Java. Developers might face more problems. Strictly speaking, there is no outright winner in implementing DockerAlt among the choices of Java, OCaml and Scala. They are all statically typed and of good flexibility, while neither of them have LXC API support. I would recommend to use Scala out of these three alternatives. The learning curve of OCaml is steeper than the curves of Java and Java-like Scala. If the developer is not familiar with OCaml, applying DockerAlt written in OCaml may be painful. In general, Scala is likely to give better performance than Java since it extends the concurrency support in the Java language.

References

[1] Batch compilation (ocamlc). `https://caml.inria.fr/pub/docs/manual-ocaml/comp.html`.

[2] Docker. `https://www.docker.com`.

[3] Docker: an insider view. `https://www.slideshare.net/jpetazzo/docker-and-go-why-did-we-decide-to-write-docker-in-go`.

[4] Docker from wikipedia. `https://en.wikipedia.org/wiki/Docker_(software)`.

[5] The java language environment. http://www.oracle.com/technetwork/java/intro-141325.html.

[6] Linux containers. https://linuxcontainers.org/lxc/.

[7] Ocaml. https://ocaml.org.

[8] Ocaml documentation. http://caml.inria.fr/pub/docs/manual-ocaml/.

[9] Ocaml portability. https://ocaml.org/learn/portability.html.

[10] Scala and go: A comparison of concurrency features. https://www.cs.colorado.edu/~kena/classes/5828/s12/presentation-materials/smithbrentgibsonleon.pdf.

[11] Scala garbage collection. http://stackoverflow.com/questions/19486562/scala-garbage-collection.

[12] Scala language specification. https://www.scala-lang.org/files/archive/spec/2.12.

[13] Scala: Object-oriented meets functional. https://www.scala-lang.org.

[14] Type inference. http://www.cs.cornell.edu/courses/cs3110/2016fa/l/17-inference/notes.html.

[15] Type inference from wikipedia. https://en.wikipedia.org/wiki/Type_inference.