UCLA Computer Science 131 DockAlt Container Report

Qingwei Lan
University of California, Los Angeles

Abstract

Docker is an open platform for building, shipping and running distributed applications. Docker is implemented in a relatively young programming language, Go. Also there is only a single source for it. Due to these problems, it may be unstable. In the following report, I will examine alternate programming languages, including Java, Python, and Rust, that are suitable for implementing DockerAlt, an alternative to Docker.

1 Introduction

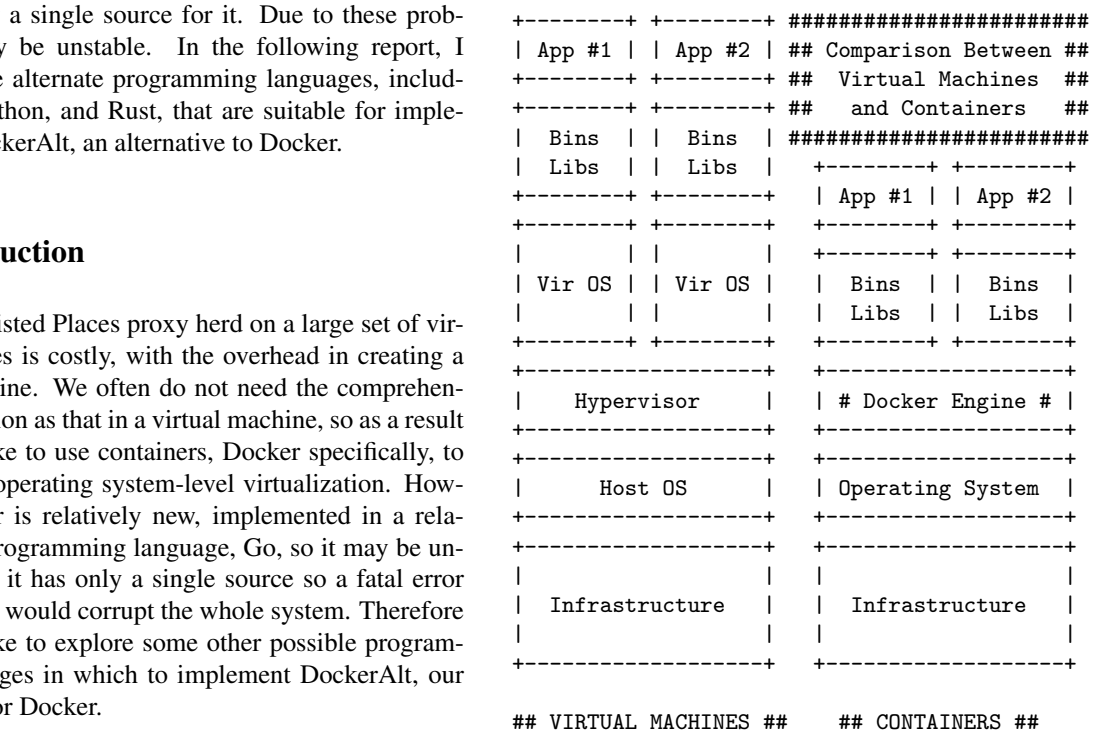
Running Twisted Places proxy herd on a large set of virtual machines is costly, with the overhead in creating a virtual machine. We often do not need the comprehensive abstraction as that in a virtual machine, so as a result we would like to use containers, Docker specifically, to accomplish operating system-level virtualization. However, Docker is relatively new, implemented in a relatively new programming language, Go, so it may be unstable. Also it has only a single source so a fatal error in the source would corrupt the whole system. Therefore we would like to explore some other possible programming languages in which to implement DockerAlt, our alternative for Docker.

2 Introduction to Containers

A container is a layer of virtualization above the operating system. Containers offer an environment extremely similar to one from a virtual machine but without the overhead that comes with running a separate kernel and simulating all the hardware. This is achieved through a combination of kernel security features such as namespaces, mandatory access control and control groups.

2.1 Containers vs. Virtual Machines

Shown below is a structural comparison between virtual machines and containers.



As shown above, in virtual machines, we run an entire guest operating system above the host operating system, involving an extra level of complete high-level abstraction. This requires extra resources, such as CPU and RAM, which is not desirable in our case when we do not need so complicated an abstraction. Using a container, in our case, Docker, we do not need an extra guest operating system. The 'Docker Engine' runs atop the host operating system and all we need to do is make sure the dependencies are met and then we can run our apps.

3 Docker and Go

Docker is implemented in Go, a programming language developed in 2007 primarily by Google engineers Robert Griesemer, Rob Pike, and Ken Thompson. This makes both Go and Docker relatively new technology. Similarly, both have gained massive popularity, although Go primarily gained popularity due to Docker.

Go is designed primarily for systems programming. It is an imperative, compiled, and statically typed language in the tradition of C/C++. It also has garbage collection, various safety features and CSP-style concurrent programming features.

Go has a lightweight design, since the designers shared a dislike of the complexity of C++. It also had a simple syntax, and its type inference makes it look like dynamic languages with dynamic typing, although it is in fact strongly typed. Consider the following code comparison between variable declaration and initialization.

```
Go                C/C++
-----------------------------------------
i := 3            int i = 3;
-----------------------------------------
s := "Go!"        const char *s = "Go!";
-----------------------------------------
```

Automatic type inference makes the language less verbose and also makes it harder for programmers to mess up types. This feature, type inference, exists in many modern programming languages, such as Swift, developed by Apple.

Go also supports the notion of tuples, in which functions in Go can return tuples, making it easier to check for errors. The common usage is for a function to return a `result, err` pair, and the caller could simply check the returned `err` variable for errors or simply discard it.

Go supports self-defined types and a number of builtin types, including complex numbers. It supports pointers but does not support pointer arithmetic.

Go supports 'classes' in the notion of 'interfaces'. Unlike a 'class' that inherits 'free' methods and variables, an interface inherits 'obligations'. When an type declares that it implements an interface, then it will have to fullfil the requirements of the interface (e.g. implement all methods in the interface). In this sense, Go is also similar to Swift. A type can implement multiple interfaces. For this reason, functions can be declared 'on' types, making it a function. For example, we could declare a `toString` function on a `int64` as follows

```
func (v int64) toString(f format) string { ... }
```

The `(val int64)` shows that we have declared the function on a `int64`. The `(f format)` is the argument to this function, where '`format`" is a self-defined type. The '`string`" at the end indicates that this method returns a string.

Go also supports parallel programming in the form of 'goroutine's and 'channel's. Goroutines are not data race free, so it is still up to the programmer to ensure that data shared across multiple threads are synchronized safely.

The authors of Go also provided a few useful tools for developing with Go, including

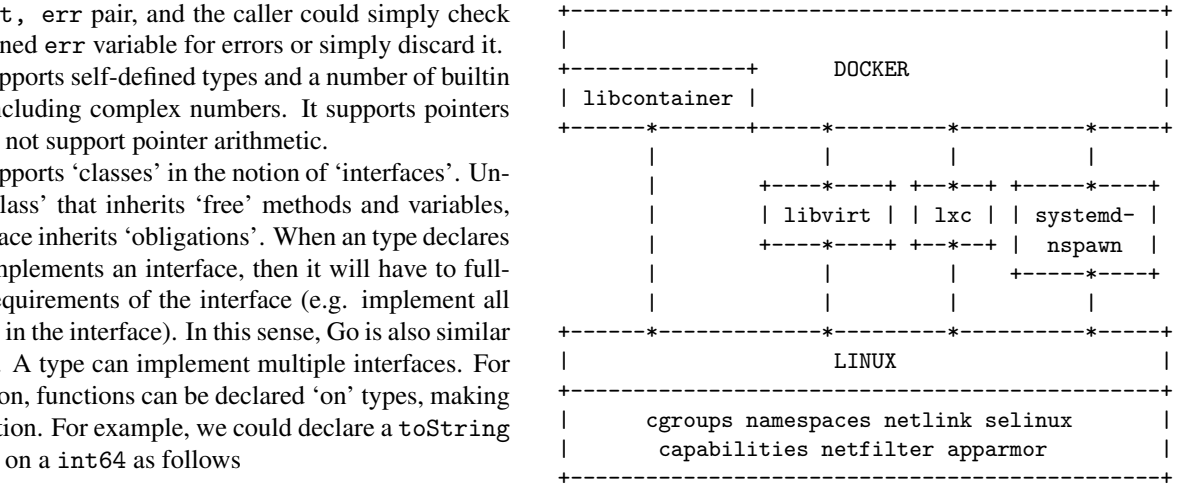
```
-----------------------------------------------
go build   install dependencies, build program.
go run     shortcut for running a Go program.
go get     retrieve and install remote packages.
go fmt     format Go code in Go style.
godoc      display the documentations of Go.
-----------------------------------------------
```

In summary, Go is statically compiled, supports strong typing with type inference, supports good lightweight parallel programming, and has gained community support worldwide, therefore having many libraries and packages. Compiling a Go program would not requrie you to worry about the dependencies as you would have to in C/C++ with all the makefile business. These attributes have led the Docker authors to choose Go as the primary language in which to implement Docker.

4 Alternate Languages for DockerAlt

We will consider the following three languages that may be good candidates for developing DockerAlt: Java, Python, and Rust.

The main structure of Docker 0.9 looks like the following, where '`libcontainer`' has been introduced in addition to the original '`LXC`'.



4.1 Java

Java can be viewed as a compiled language since it compiles source code into Java bytecode and runs it on a

JVM. Because of this, it is faster than Python, which is an interpreted language. Java is a mature language with great community support worldwide. Libraries and packages in Java enable developers to work with all sorts of supported and stable code.

Java also has libraries that support parallel programming, which, if compared to Python, are lightweight. It also comes with a garbage collector, making it mostly memory-safe. Another great thing about Java is that it is platform-independent, very much like Go, so we do not need to worry too much about shipping platform-dependent code.

However, Java is heavy compared to Python and Go, which makes it verbose for developing DockerAlt from scratch and also hard to understand. One example would be that it does not support type inference or dynamic typing, making it verbose when declaring varibles. Java is hard to start with, especially with all its supported libraries, and we have to admit, without those libraries, there is not point to learn Java.

Another disadvantage is that Java does not have native support for `LXC`, which Docker is built upon. But currently with Docker 0.9, it introduced '`libcontainer`', which is written in pure Go, and does not depend solely on `LXC` anymore. Either way, Java does not have support for either of these technologies, so it would be difficult to develop DockerAlt in Java.

4.2 Python

Python is in many ways similar to Go, and has many advantages over Go as well. It is a suitable candidate for developing DockerAlt, for it has dynamic typing, supports parallelism, and has a garbage collector. The only disadvantage is that is a interpreted langauge, therefore it may be slower than Go, but modern Python programs are also gaining speed due to community support and new tools.

Python is easy to learn and easy to code and debug. In the previous project, we showed that we could achieve a simple server in about 200 lines of code. Apart from this, Python also has native support for `LXC`, open-sourced on Github, known as '`python2-lxc`'. This makes Python a valid candidate for implementing DockerAlt.

4.3 Rust

Rust, funded by Mozilla Research, is a systems programming language focused on three advertised goals: safety, speed, and concurrency. By this, Rust seems to be a valid candidate for implementing DockerAlt.

Rust does not have a garbage collector, but does memory management statically at compile time. This greatly increases performance but also introduces extra, though manageable complexity to the language. Rust uses the notion of 'ownership'. When we allocate an object, the variable we assign it to is the 'owner' of the object, and as long as the owner stays in scope, the object will exist.

Rust also supports simple parallelism with nice documentation. This is one of the features we would need in building DockerAlt.

Rust developers have also developed a tool for managing Rust projects, called 'Cargo'. Although this is still in pre-release, it is growing stably and people have converted to starting projects with Cargo. For large projects such as DockerAlt, Cargo would prove useful.

However, Rust is also a relatively new programming language, with the first stable version released in May 2015. Since we are already worried with Go not being stable enough, it does not make too much sense to use Rust as it is a even newer language than Go. Also, Rust does not have support for `LXC`, which would make DockerAlt hard to implement.

5 Conclusion

With the analysis of the languages Java, Python, and Rust in the report above, we can see that both Java and Rust are not valid candidates for implementing DockerAlt. Java introduces extra complexity and requires an expert programmer to start the project. Rust is much newer than Go, and support worldwide does not exceed that of Go. Also both of these languages do not have native support for `LXC`, which makes these two languages bad candidates for rewriting Docker.

However, Python possesses the properties of implementing DockerAlt. It has simple syntax, lots of community support, with many libraries and packages. It has automatic memory management in a combination of 'automatic reference counting' and 'garbage collector'. It has dynamic typing, easy parallel programming support, and most of all, native support for `LXC` in both Python 2 and Python 3. This makes Python the best alternative for implementing DockerAlt.

References

[1] Asay, Matt. "10 things you should know about Docker", http://www.techrepublic.com/blog/10-things/10-things-you-should-know-about-docker/, 3 Dec. 2015.

[2] Banerjee, Tobby. "Understanding the key differences between LXC and Docker", https://www.flockport.com/lxc-vs-docker/, 2 Dec. 2015.

[3] Baukes, Mike. “Docker vs LXC”, https://www.scriptrock.com/articles/docker-vs-lxc, 1 Dec. 2015.

[4] Bryant, Christian. “LXC: Move Over Virtual Machines, Here Come Containers”, http://www.tomsitpro.com/articles/lxc-linux-containers-docker,1-1904.html, 2 Dec. 2015.

[5] Gulati, Shekhar. “Day 21: DockerThe Missing Tutorial”, https://blog.openshift.com/day-21-docker-the-missing-tutorial/, 2 Dec. 2015.

[6] Hemel, Zef. “Docker: Using Linux Containers to Support Portable Application Deployment”, http://www.infoq.com/articles/docker-containers/, 2 Dec. 2015.

[7] Hykes, Solomon. “Docker 0.9: introducing execution drivers and libcontainer”, http://blog.docker.com/2014/03/docker-0-9-introducing-execution-drivers-and-libcontainer/, 3 Dec. 2015.

[8] Litt, Steve. “Newbie’s Overview of Docker”, http://www.troubleshooters.com/linux/docker/docker-newbie.htm, 3 Dec. 2015.

[9] Petazzoni, Jrme. “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, 3 Dec. 2015.

[10] Petazzoni, Jrme. “PaaS under the hood, episode 1: kernel namespaces”, http://web.archive.org/web/20150326185901/http://blog.dotcloud.com/under-the-hood-linux-kernels-on-dotcloud-part, 3 Dec. 2015.

[11] Swan, Chris. “Docker drops LXC as default execution environment”, http://www.infoq.com/news/2014/03/docker_0_9, 3 Dec. 2015.

[12] Yegulalp, Serdar. “4 reasons why Docker’s libcontainer is a big deal”, http://www.javaworld.com/article/2363024/big-data/4-reasons-why-dockers-libcontainer-is-a-big-deal.html, 3 Dec. 2015.