Containers and DLL Hell

Adrian Kosmaczewski

2022-11-11

Back in the 1990s, shared libraries were all the rage. Instead of having to ship a 20 MB `*.exe` file to your customer in various floppy disks, you could cut some code out, put it in a set of `*.dll` files, and reuse that code across all your products. Every vendor would then install lots of DLL files in your system, and they would be reused by other apps from the same vendor.

Why did they do that? Well, storage space was expensive back then. An internal 1 GB hard disk in 1995 costed me around a thousand Swiss Francs, and it filled fast with all those apps (well, I also had Windows 3.1 and OS/2 Warp installed simultaneously.) So reusing code in DLLs appeared, at least at first sight, as a valuable mechanism to reduce bloat for software vendors, and to avoid repeating themselves.

The truth is that those DLLs multiplied themselves like bacteria, and led to a concept that made headlines back in 1995: DLL Hell. The concept was so pervasive and problematic that it got its own Wikipedia entry[1].

Talk about being a celebrity.

Even Microsoft couldn't cope with the DLL explosion; apps would crash at runtime because they would load DLLs with the same name, but with different versions of the same functions inside. It was really a mess.

The most egregious example of this mess was Visual Basic itself, and its suite of `VBRUN300.DLL` libraries, of which it seemed there were gazillion versions in shareware collections everywhere. Which one to use? Well, you're out of luck; try them one by one until your game ~~runs~~ doesn't crash anymore.

The best way to prevent DLL Hell was (still is) to link binaries with libraries at build time. But of course, if your storage is expensive, well, it's not the best way to do things, because the final executable will be more bulky at the end. But, look ma! No DLL loading at runtime! Static binaries are the best.

But this is only valid for compiled languages, really. If your app is written in PHP, Python, Ruby, Perl, Lua, or (more recently) JavaScript, well, you'll need to install a few gigabytes of packages and runtimes for them to run. `README` is your friend, and then `npm install` or `pip install` or `gem install` until everything works.

We're in 2022 now, almost 30 years later, and statically built binaries have won, and even for scripting languages! They are called containers now; Docker

[1] https://en.wikipedia.org/wiki/DLL_Hell

containers, pods, what have you, and they encapsulate[2] not only the final executable and its libraries, but also any other piece of runtime code they might need. Like the whole .NET framework, or the PHP runtime, or a complete version of Ruby and Ruby on Rails[3] including all the gems your app needs to get things done.

It does not matter anymore if your binary is *actually* statically linked, though; apps written in Ruby, Python, PHP, JavaScript, Lua and other similar languages are just interpreted on the fly by the runtime installed within the same container.

The downside is that with those languages your final container image could easily stretch into the gigabytes… not very convenient if you want to run many copies of those containers in the same load-balanced Kubernetes[4] deployment. Oops.

This all means that compiled languages are making a nice comeback right now; Go[5], Rust[6], D[7], C++[8], .NET[9], Crystal[10], C[11], they can all be used to generate a small self-contained executable, and with it, a very small final container image.

Specify in your `Dockerfile` a base image `FROM scratch`, copy your binary, and your container image is now just 15 MB big. Push it, share it[12], pull it, and reuse it. And apparently .NET 7 (the latest version) includes a new AOT feature[13] that makes really small native binaries. Finally. Oh, and you can even `dotnet publish` directly as a container.

Talk about convenient. Make small container images, people.

[2] /blog/fortune-apps/
[3] /blog/the-technical-news-of-the-day/
[4] /blog/kubernetes-for-non-technical-readers/
[5] /blog/thoughts-about-googles-go-programming-language/
[6] /blog/first-web-app-in-rust/
[7] /blog/d-or-what-go-may-have-been/
[8] /blog/blow-your-mind/
[9] /blog/a-linker-for-joel/
[10] https://akos.ma/blog/crystal-is-a-surprise/
[11] /blog/how-knowing-c-and-c-can-help-you-write-better-iphone-apps-part-1/
[12] /blog/reusing-apps-between-teams-and-environments-through-containers/
[13] https://learn.microsoft.com/en-us/dotnet/core/deploying/native-aot/