What is(n't) Docker actually for?

> This article was originally published at https://gist.github.com/joepie91/1427c8fb172e07251a4bbc1974cdb9cd.

> This article was written in 2016. Some details may have changed since.

A brief listing of some misconceptions about the purpose of Docker.

Secure isolation

Some people try to use Docker as a 'containment system' for either:

- Untrusted user-submitted code, or
- Compromised applications

... but Docker explicitly does not provide that kind of functionality. You get essentially the same level of security from just running things under a user account.

If you want secure isolation, either use a full virtualization technology (Xen HVM, QEMU/KVM, VMWare, ...), or a containerization/paravirtualization technology that's explicitly designed to provide secure isolation (OpenVZ, Xen PV, *unprivileged* LXC, ...)

"Runs everywhere"

Absolutely false. Docker will not run (well) on:

- Old kernels
- OpenVZ
- Non-*nix systems (without additional virtualization that you could do yourself anyway)
- Many other containerized/paravirtualized environments
- Exotic architectures like MIPS

Docker is *just* a containerization system. It doesn't do magic. And due to environmental limitations, chances are that using Docker will actually make your application run in *less* environments.

No dependency conflicts

Sort of true, but misleading. There are *many* solutions to this, and in many cases it isn't even a realistic problem.

- **Compiled languages:** Just compile your binary statically. Same library overhead as when using Docker, less management overhead.
- **Node.js:** Completely unnecessary. Dependencies are *already* local to the project. For different Node.js versions (although you generally shouldn't need this due to LTS schedules and polyfills), nvm.
- **Python:** virtualenv and pyenv.
- **Ruby:** This one might actually be a valid reason to use *some* kind of containerization system. Supposedly tools like `rvm` exist but frankly I've never seen them work well. Even then, Docker is probably not the ideal option (see below).
- **External dependencies and other stuff:** Usually, isolation isn't necessary, as these applications tend to have extremely lengthy backwards compatibility, so you can just run a recent version.

If you *do* need to isolate something and the above either doesn't suffice or it doesn't integrate with your management flow well enough, you should rather look at something like Nix/NixOS, which solves the dependency isolation problem in a *much* more robust and efficient way, and also solves the problem of state. It does incur management overhead, like Docker would.

Magic scalability

First of all: you probably don't *need* any of this. 99.99% of projects will never have to scale beyond a single system, and all you'll be doing is adding management overhead and moving parts that can break, to solve a problem you never had to begin with.

If you *do* need to scale beyond a single system, even if that needs to be done rapidly, you probably *still* don't get a big benefit from automated orchestration. You set up each server once, and assuming you run the same OS/distro on each system, the updating process will be basically the same for every system. It'll likely take you more time to set up and manage automated orchestration, than it would to just do it manually when needed.

The only usecase where automated orchestration *really* shines, is in cases where you have high *variance* in the amount of infrastructure you need - one day you need a single server, the next day you need ten, and yet another day later it's back down to five. There are extremely few applications that fall into this category, but even if your application does - there have been

automated orchestration systems for a long time (Puppet, Chef, Ansible, ...) that don't introduce the kind of limitations or overhead that Docker does.

No need to rely on a sysadmin

False. Docker is not your system administrator, and you still need to understand what the moving parts are, and how they interact together. Docker is *just a container system*, and putting an application in a container doesn't somehow magically absolve you from having to have somebody manage your systems.

Revision #1
Created 11 December 2024 18:19:27 by joepie91
Updated 11 December 2024 18:21:06 by joepie91