

Docker: Automated and Consistent Software Deployments



infoq.com/news/2013/03/Docker

Abel Avram

March 27, 2013



Write for InfoQ

Feed your curiosity. Help 550k+ global senior developers

each month stay ahead. [Get in touch](#)

[dotCloud](#), a PaaS provider, has open sourced [Docker](#), a key component of their platform.

Docker is a [Linux Container \(LXC\)](#) technology augmented with a high level API providing a lightweight virtualization solution that runs Unix processes in isolation. It provides a way to automate software deployment in a secure and repeatable environment.

Docker uses the concept of a Standard Container which contains a software component along with all its dependencies - binaries, libraries, configuration files, scripts, virtualenvs, jars, gems, tarballs, etc. – and can be run on any x64-bit Linux kernel that supports [cgroups](#). Such containers can be deployed on a laptop, on a distributed infrastructure, in the cloud, etc., preserving its environment, making it appropriate for a broad range of uses: continuous deployment, web deployments, database clusters, SOA, etc., as [Mike Kavis explained on his blog](#):

The use case that was relevant to me, the application guy, is to use Docker to streamline a continuous delivery process. In every place that I have worked in my career, from the mainframe days, to the client server days, to the cloud days, getting the different environments in sync and successfully testing applications has been a nightmare. When code moves from Dev to QA to Stage to Prod, no matter how good or bad our processes were these environments were NEVER the same. The end result was always a hit in the quality of a production release. "It worked in test" became the most shrugged off phrase since "the check is in the mail".

With Continuous Delivery (CD), the *entire environment* moves with the code from Dev to QA to Stage to Prod. No more configuration issues, no more different systems, no more excuses. With CD, if it didn't work in Prod it didn't work in Test. With Docker, I can see writing scripts to automate the CD process. I can see gains in speed to market because of how quickly new environments can be created without dealing with all of the setup and configuration issues.

Solomon Hykes, CEO of dotCloud, [demoed Docker at PyCon](#), explaining that it's a repeatable lightweight virtualization solution because "it's isolated at the process level and it has its own file system". The API enables system administrators to execute a number of operations on containers: start, stop, copy, wait, commit, attach standard streams, list file system changes, etc.

Some of Docker's [main features](#) are:

- File system isolation: each process container runs in a completely separate root file system.
- Resource isolation: system resources like CPU and memory can be allocated differently to each process container, using cgroups.
- Network isolation: each process container runs in its own network namespace, with a virtual interface and IP address of its own.
- Copy-on-write: root file systems are created using copy-on-write, which makes deployment extremely fast, memory-cheap and disk-cheap.
- Logging: the standard streams (stdout/stderr/stdin) of each process container are collected and logged for real-time or batch retrieval.
- Change management: changes to a container's file system can be committed into a new image and re-used to create more containers. No templating or manual configuration required.
- Interactive shell: docker can allocate a pseudo-tty and attach to the standard input of any container, for example to run a throwaway interactive shell.

So far, Docker has been tested with Ubuntu 12.04 and 12. 10, but it should be working with any Linux 2.6.24 or later, according to dotCloud. It can also be installed on Windows or Mac OS X via [VirtualBox](#) using [Vagrant](#). Docker was written in Go, and uses Linux [cgroup](#) and [namespacing](#), [AUFS](#) – file system with copy-on-write capabilities-, and [LXC](#) scripts.