What is Docker and why use it?

Docker is the “poster child” of the containerization movement but is it here to stay? Despite the buzz, many application developers are still undecided about using containers in production. In this article we will talk about:

- the need for containerization
- Docker’s benefits and best use cases
- containerization alternatives in the market
- going into production with Docker in enterprise-grade environments

What are containers?

The advent of web based software, services and platforms along with commoditization of computing services means that scaling has become of paramount importance. Even more, new paradigms in software development – agile techniques, which shorten the path between the developer and the production environments, also contributed to an increase in adoption among enterprises and startups alike. These new development practices and a new set of requirements, in terms of continuous integration and continuous deployment meant that there was a proper seeding ground for ideas like containerization.

Containerization tools enable ‘immutability’ in the infrastructure: container apps that are built ‘on the spot’ and ‘in-place’ with the same configuration and same dependencies as the original authors intended. This is one of the main reasons (or perhaps even the main driving force) of their rapid adoption and increased usage. These tools allow application developers to focus on the application instead of focusing on the infrastructure while bringing versioning to application image distribution. It even goes as far as bringing concepts that “traditionally” belonged to a different area: you can do pulls, pushes and commits on Docker images, concepts borrowed from Source Code Management Software (like Git and Mercurial).
To get any confusion out of the way, Docker refers both to an open-source project (https://github.com/docker/docker) as well as the company behind it – Docker Inc. (formerly dotCloud). dotCloud was a PaaS player who built Docker for internal use. Once they realized its potential, they pivoted and focused exclusively on developing Docker. The importance of the project in the greater ecosystem was also recognized by most of the industry, thus allowing Docker

Inc. to raise 40M$ in the most recent funding round. Following the pivot dotCloud (the PaaS offering of the company) was ‘relocated’ to Berlin-based cloudControl.

Docker is just one of the available containerization offerings and, as many similar projects, relies on kernel features that (in Linux) have been available for more than 6 years (since around 2007). However, as with other technologies (see e.g. boom in smartphone sales with the introduction of iPhone) Docker has only put a user (and developer) friendly interface around pre-existing components. The same concepts appeared 2-3 years earlier in Solaris OS (build 51 in Solaris 10), known as Solaris Containers w/ Solaris Zones. While initially Docker was a wrapper for LXC (LinuX Containers), nowadays it manages libcontainer – a unified interface for cgroups and kernel namespaces. The gist is: containerization through kernel spaces enables multiple user spaces, while the users ‘occupying’ these spaces have no – okay, you got me, they have a little – knowledge that other users exists.

Docker vs. Virtualization

While there are a ton of similarities there are also at least two tons of differences between the two. The most important difference is that, in Docker, there’s no overhead introduced by the need to emulate an entire OS for the virtual machine. As such, Docker provides better performance (in terms of speed and capacity) when for custom applications in a container, compared to virtual machines, provided that the VM host and LXC host have the same hardware characteristics. Another important difference is that, with Docker, the host and the container have to run the same type of OS.

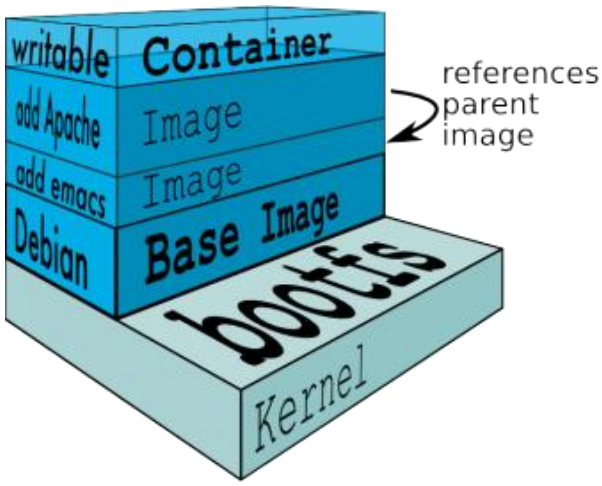
A less known fact about virtualization is that it has appeared close to 50 years ago at IBM (where else?), in their CP-40 mainframe and was later introduced into several other product lines (such as iSeries and zSeries). Just like a plethora of other technologies, the continuous demand for better performance, better management and easier deployment and the advances in hardware and software made the concept of virtualization reach the pinnacle of its evolution in the recent years.

Both containers and Virtual Machines address the same problem – isolation and control of components of an application – but this is achieved in different way as containers give up some of the isolation for a more efficient usage of the (host) system resources.

Why developers love Docker

Docker 101

Docker relies for its execution environment, on features in the host's kernel – LXC. But it also needs filesystem support in the so called UFS (Union File System).



Docker Filesystems Multilayer

Early versions of Docker relied on AuFS, but more recent releases have adopted a 'neutral' approach, with a preference for devicemapper. This use of a copy-on-write FS is actually what makes Docker look like Git (and Docker Hub like GitHub).

Docker's two main concepts are Containers and Images, where Containers are, well, containers running the application and Images are the Containers-at-Rest (i.e. saved container state). To make an analogy with class concepts in OOP, Images are class definitions, while Containers are class instances at run-time.

All in all there are just 28 commands that the tool called Docker (self-entitled "self-sufficient runtime" for Linux containers) understands, but they wrap all the mentioned capabilities – including the control (such as start/stop a container) and the meta-management (such as push/pull to/from the repository).

For instance starting a container running Redis is as simple as*:

```
docker pull dockerfile/redis

docker run -d --name redis -p 6379:6379 dockerfile/redis
```

(*note that there are other Redis 'templates' as well)

This will allow one to subsequently use

```
<container_host_ip>:6379
```

to connect to the Redis server. To get a feeling of what Docker is like just visit https://www.docker.com/tryit/ to access a browser based interactive tutorial.
The latest version of Ubuntu Linux (14.04 LTS) comes with activated Docker support. It still needs to be installed, though that's just an 'apt-get install docker.io'.

Docker Benefits

The roles that have most to benefit from Docker are those at the two ends of the development cycle (developer and dev-ops/sysadmins), but it also improves the lives of everyone else in between (like QA staff).

Docker's most appreciated benefits are:

- **application-centric**: the incremental improvement that Docker brings on LXC is focus on deployment of application vs deployment of machines
- **portability**: once published, any image will yield the exact same result wherever it runs
- **versioning**: much like Git, allows commit/push and pulls on existing images, verifying differences from one commit to the other
- **automation**: there are tools that allow a machine, once running, to reach a specific state
- **sharing**: through the use of Docker Hub anyone can build on existing machines or make available their images for others
- **reusability**: an image can be 'fork'-ed for two different purposes.

Beyond Docker: are containers here to stay?

Ecosystem

Docker in itself, but also components and concepts popularized by it, are at the heart of a multitude of projects which try to build on it. Just as Docker put a friendly face around LXC, people and companies around the world try to use it to put friendly faces around other connected areas.

For instance a consortium comprising of the technology behemoths Google, Microsoft and IBM as well as smaller companies, such as CoreOs, RedHat, Mesosphere and even the apparent Docker competitor VMWare, is supporting a project called Kubernetes (open-sourced earlier this year by Google) which tries to "dockerize" what happens above-and-beyond the container (i.e. defining logical components that run in different applications but are managed in a cluster as a single entity). Here is a non-exhaustive list of related (and sometime competitive) containerization projects:

● https://github.com/GoogleCloudPlatform/kubernetes – container cluster management.

● https://github.com/progrium/dokku – Docker powered mini-Heroku. The smallest PaaS implementation you've ever seen

● http://deis.io – open source PaaS that makes it easy to deploy and manage applications on your own servers

● https://github.com/signalfuse/maestro-ng – deployment and control of complex, multi-host environments using Docker containers possible and easy to use

● http://www.serfdom.io – Serf is a tool for cluster membership, failure detection, and orchestration that is decentralized, fault-tolerant and highly available

● https://mesosphere.github.io/marathon – Docker plus Mesosphere provides an easy way to automate and scale deployment of containers in a production environment. Marathon enables REST API for Mesos

● https://github.com/newrelic/centurion – A deployment tool for Docker. Takes containers from a Docker registry and runs them on a fleet of hosts with the correct environment variables, host volume mappings, and port mappings

● https://github.com/mesosphere/deimos – Mesos containerizer hooks for Docker

● http://www.fig.sh – Fast, isolated development environments using Docker (acquired by Docker Inc)

● http://shipyard-project.com – manage Docker resources including containers, hosts and more

● http://www.projectatomic.io

Is Docker ready for production? All the signs seem to point to YES.

Probably the largest user of containers (based on LXC, not necessarily Docker) in production is at Google, where "everything, from Search to Gmail, is packaged and run in a Linux container. Each week we launch more than 2 billion container instances across our global data centers, and the power of containers has enabled both more reliable services and higher, more-efficient scalability" (quote from Google Cloud Platform Blog).

Furthermore, as the decision power shifts (not entirely, but in significant amounts anyway) to the developers (see e.g. The New Kingmakers by RedMonk's Stephen O'Grady) and as Docker is a tool built by developers (primarily) for developers (although Docker Inc. only had approx 30 employees at the time of releasing 1.0, this release came through the shared effort of more than 460 people) the project stroke a chord with a lot of people leading to almost 3M downloads. Considering that Docker is not a consumer application, this is in itself a remarkable achievement.