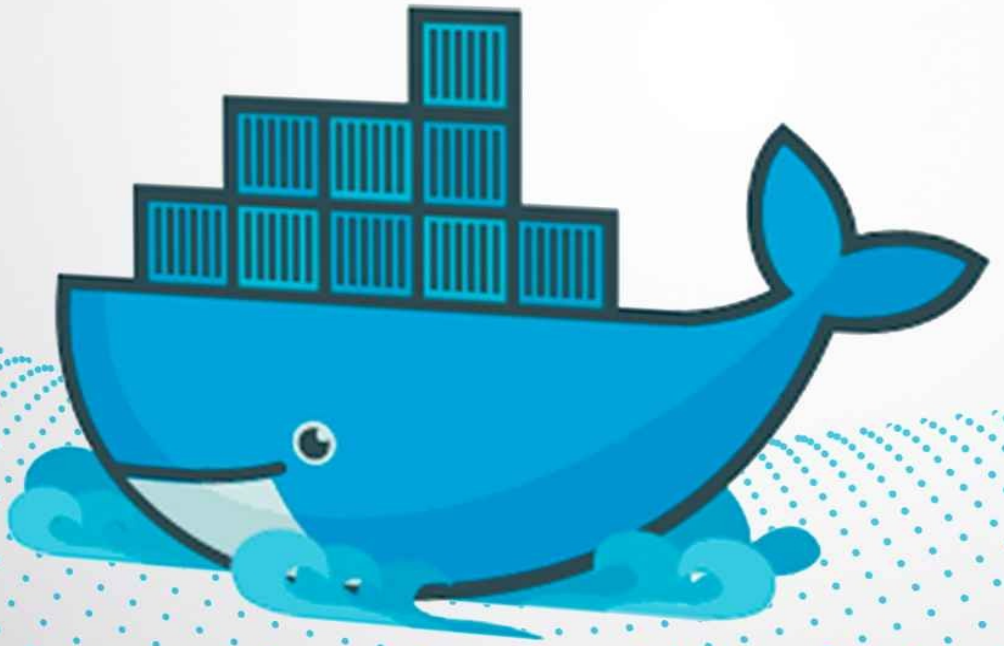
DOCKER

WHAT YOU NEED TO KNOW ABOUT CONTAINERS



JOHN SYMONDS


docker

What You Need To Know About Containers

Disclaimer / Legal Notice

The content within this publication are purely the express opinions of the Author. Although the Author has made every effort to ensure that the information in this publication was correct at time of writing, the Author and Publisher do not assume and hereby disclaim any liability to any party for any loss, damage, or disruption caused by errors or omissions, whether such errors or omissions result from negligence, accident, or any other cause.

The Author and Publisher do not endorse, recommend, control, makes no representation or warranty of any kind with respect to any external sources or entities referenced within this publication. The Author and Publisher do not assume and hereby disclaim any liability to any party for any loss, damage, or disruption attributed to any external sources or entities.

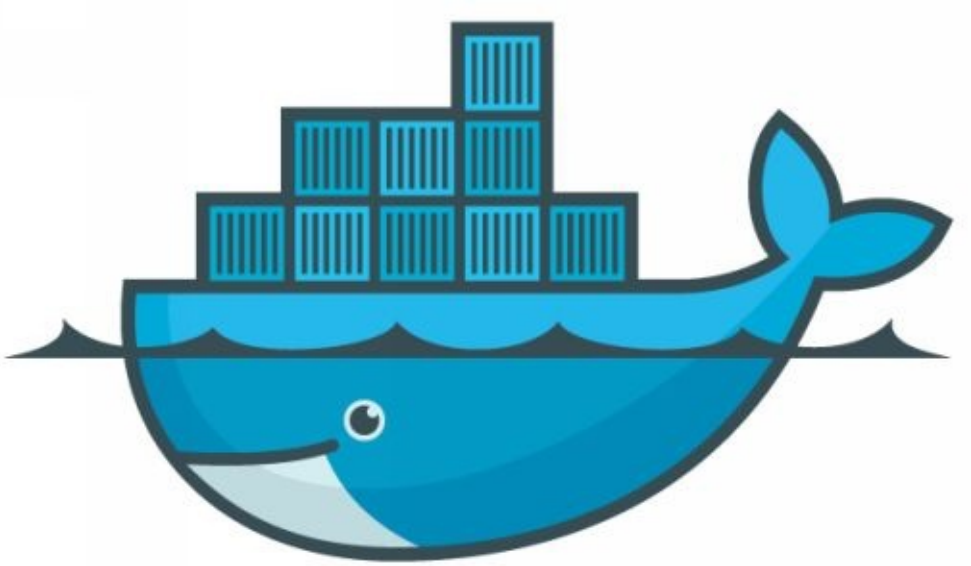
Table of Contents

Introduction

Docker has exploded onto the scene in recent times, fast becoming the de-facto standard for developers and system administrators for packaging, deploying and running applications.

Docker greatly simplifies the Developer Operations (DevOps) workflow by enabling greater automation to promote consistency and reliability.

This book will give you a fundamental understanding of Docker and container-based virtualization.



Chapter 1

The Basics: Virtualization

To understand the problem that Docker solves, first we need to take a step back and understand the basics of virtualization and the problems we currently face.

Virtualization is the process of creating a software-based representation of physical hardware such as a server, network router, storage infrastructure etc. By representing these physical devices using virtual bits, virtualization provides significant cost savings and a higher degree of scalability. Infrastructure can literally be scaled up and down at the click of a button! For this reason the adoption of virtualization technologies has grown exponentially in recent times, in particular thanks to the rapid uptake of cloud computing and the drive toward *microservice* application architectures.

Until recently, virtual machine based virtualization had assumed to be the only way to isolate applications running on a server. Virtual machine based virtualization involved virtualizing an entire operating system - each virtual machine requiring its own operating system much like a physical server. In recent times, container-based virtualization has gained huge following as a leaner and faster alternative, most notably with the open source product Docker.

Microservices vs Monoliths

One of the strongest drivers for the adoption of Docker is the movement toward microservices application architectures.

Applications were traditionally developed as a single monolithic process that would handle all core functions of the application. This approach lead to large and overly-complex applications that were hard to maintain and did not scale particularly well.

Microservices architecture is an approach to developing a single application that is comprised of several smaller standalone services that are deployed and managed separately from one another. Each component is dedicated to completing a common set of tasks, resulting in a more flexible and easier to manage application. This approach lends itself well to distributed applications, especially important in today's age of cloud computing.

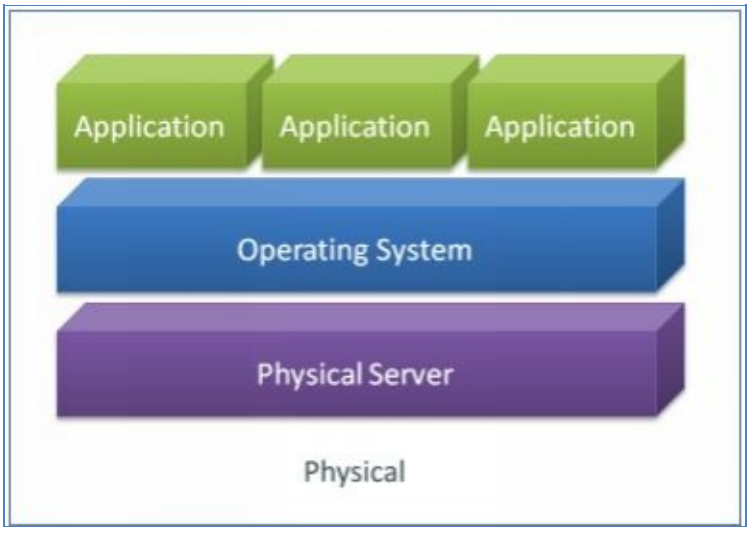
Chapter 2

What is Docker

D*ocker Inc.* is the company behind the *Docker* open source platform allowing developers and system administrators to build, ship and run distributed applications with ease.

Docker is a container-based virtualization technology allowing applications to be packaged and hosted within highly portable, self-contained environments called containers, each container sharing the operating system kernel of the underlying host operating system.

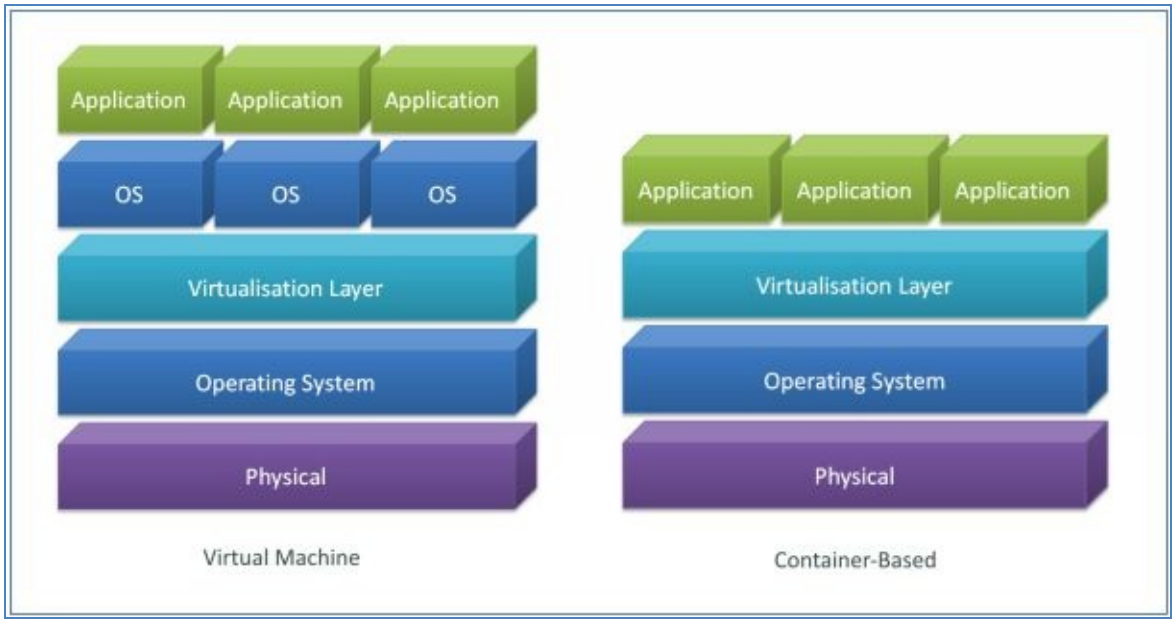
Traditionally, physical servers could only run a single operating system at a given time, for this reason, most physical servers tend to be under-utilized; on average operating at only 15 percent capacity. This is terribly wasteful and resulted in server sprawl and increased server management complexity.



There has been a significant uptake of virtual machine based virtualization technologies to improve efficiency, increase scalability and lower costs of IT spend. Virtual machines allow a single host to run applications within an isolated, sandbox environment, each containing their own operating system.

Although the concept of containers has existed for some time (including *chroot* on the Unix platform, Open VZ and Linux *lxc* containers), container-based virtualization has gained significant traction in recent times. Rather than virtualizing an entire operating system, container-based virtualization instead creates an isolated sandbox (known as a container) for an application to operate within. Multiple containers can be run on a host operating system, with each container having its own file system and running processes. Containers share the same kernel of the host operating system, removing the need for each environment to run separate operating systems - as with traditional virtual machines.

Containers are lightweight and highly portable. A developer could therefore create a container locally on his development machine and later deploy the same container to the cloud without any modification.



Benefits of this approach include:

- **Fast and consistent delivery of your applications**. Applications (and supporting dependencies) are packaged within self-contained containers that can be deployed across environments without requiring change. This provides developers with a common standardized environment to work from, removing environment to environment inconsistencies.
- **Responsive deployment and scaling**. Containers enable highly portable workflows. Containers can be deployed from a developers workstation straight to the cloud, as-is without modification. Containers facilitate dynamic workloads allowing the scale up and tear down of applications and services in near real time.
- **Better server resource utilization**. Containers are lightweight in comparison to traditional Virtual Machines. Docker promotes higher density environments by allowing more applications and services to run on a given server.

Did you know?

Docker is slang for 'dock worker'.

Dock yard workers are responsible for loading and unloading of cargo vessels that arrive in port.

In much the same way as container-based virtualization, shipping containers have revolutionized the transportation industry by providing a single standardized method to transfer cargo. Before such time, there were typically long delays to transport cargo due to a lack of global standardization; cargo handling was highly labor intensive.

CHAPTER 3

CONTAINERS VS VIRTUAL MACHINES

Containers don't necessarily replace virtual machines instead both technologies have their own strengths and weaknesses and should be used where appropriate.

Here are some key differentiators between the two:

Scalability

- Containers enable you to squeeze out more efficiency from your existing servers. They are very lightweight and therefore highly scalable. Virtual machines on the other hand incur additional overheads for the operating system.
- Containers offer better portability than virtual machines. There are a variety of virtual machine image formats available, unfortunately format compatibility varies between competing products and therefore cross compatibility of images is limited.

Productivity

- Containers are typically much simpler to setup and maintain. Each virtual machines requires the installation and maintenance of its own operating system instance (including supporting drivers), resulting in additional complexity.

Security

- Virtual machines provide a higher degree of isolation between the virtual machine and the host operating system. Virtual machines leverage hardware isolation using Intel's VT CPU and AMD's –V virtualization hardware extensions. Unfortunately hardware has not yet caught up to container-based virtualization solutions – at least at time of writing. For this reason, virtual machines are considered to be more secure, and

therefore favored in environments with high security requirements.

Performance

- According to Docker, an application running in a container will typically perform twice as fast as an application running within a virtual machine. Containers outperform virtual machines as they are considerably leaner when compared with the virtual machines. This is most evident in start-up times; virtual machines must boot a full operating system.
- Containers make better use of system resources when compared with virtual machines. Virtual machines typically consume more memory because of additional overheads, specifically the operating system.

Flexibility

- Virtual machines provide a high degree of flexibility by providing control over the underlying operating system platform. If using containers, your choice of operating system will be bound to the underlying platform of your host machine. Therefore you cannot run a Windows container on a Linux based host operating system.
- Docker containers are highly portable and can be transferred between machines with no impact to the applications that run inside them.

CHAPTER 4

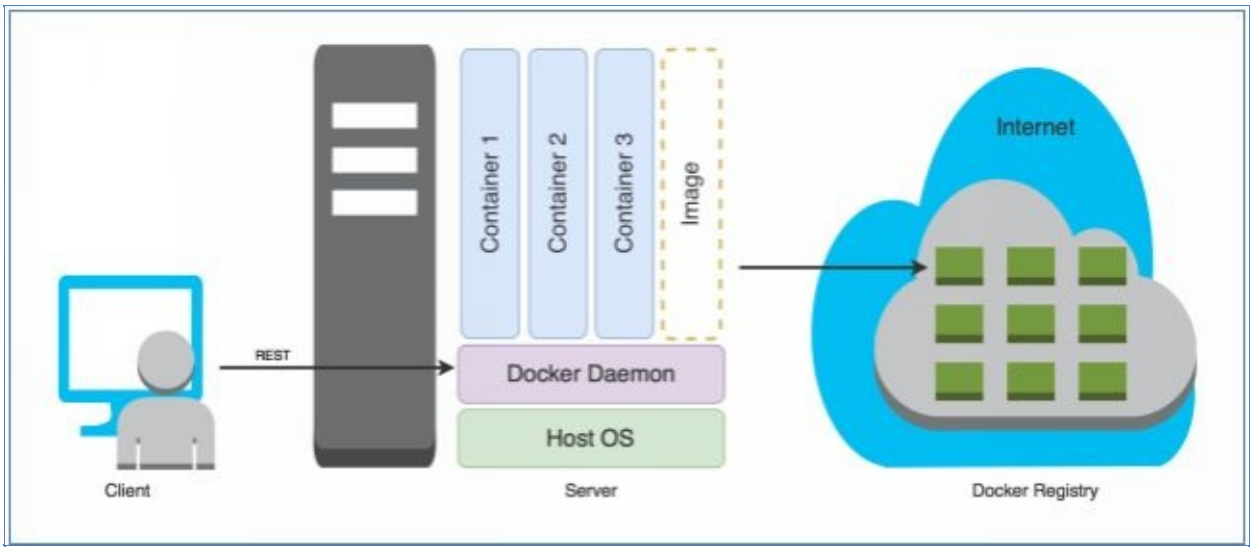
DOCKER DEEP DIVE

Overview

Docker Engine is the underlying runtime that builds and runs your containers.

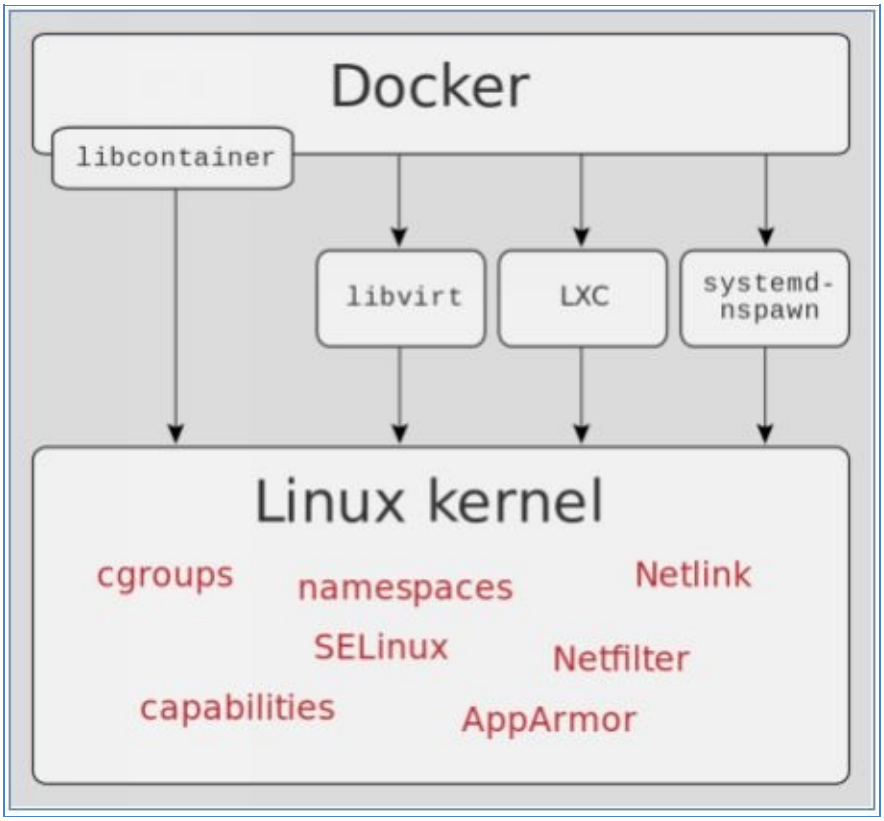
It can be broken down into 3 main components:

- **Client:** The command line user interface (CLI)
- **Server:** The Docker daemon (aka the Docker Engine) performs all the heavy lifting. It is responsible for managing all Docker objects such as images, containers, networks etc.
- **REST API interface:** Facilitates client-server communication. The CLI uses the REST API to control or interact with the server.



The operating system kernel manages the job of isolating the container processes and resource management. Although each container is running upon the same operating system platform, they each have their own file system, memory, running processes etc.

Containers provide a self-contained sandbox environment abstracting away the differences between the underlying operating system upon which they reside. This allows containers to be ported between different machines without impact to the applications that reside within them.



Containers

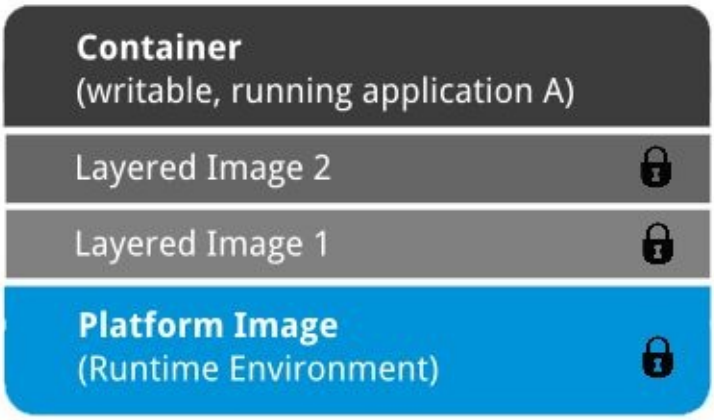
Containers provide a lightweight self-contained sandbox to run applications within.

Typically, a single application is run per container to allow maximum portability and simplify container management.

Images

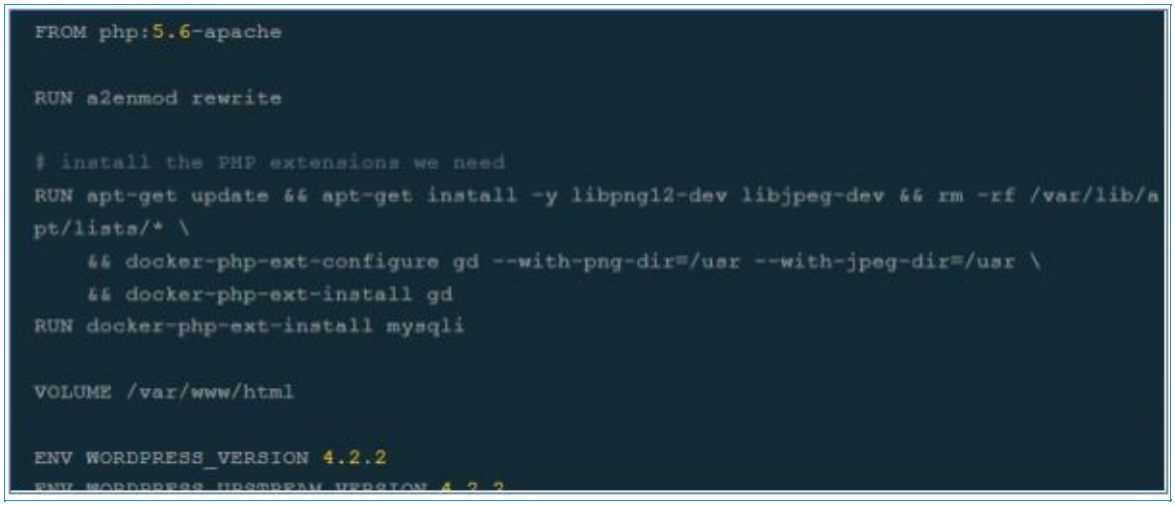
Containers are instances of Images. Images act as a template for creating new containers, or another way of looking at it, an image is to a container, as a cookie cutter is to a cookie. One cookie cutter can cut many cookies that have the same shape and properties of the cookie cutter. Likewise, you can create many containers from a single base image.

Images are composed of a series of layers, including a read-only layer containing the underlying host operating systems root file system and additional read-only layers containing snapshots of changes you've made overtime to the image. A final writable layer is created when a container is created from an image, this layer will hold any changes you make.



A Dockerfile is used to build the subsequent layers of an image. It's simply a text file containing a series of commands to execute. Changes to the base image will form the subsequent read-only layers of your image.

Here is a snippet from the official Wordpress Dockerfile. If it looks confusing at first don't worry! The main take away is that the Dockerfile uses the base PHP 5.6 Apache image and then downloads and installs the relevant packages.



```
FROM php:5.6-apache

RUN a2enmod rewrite

# install the PHP extensions we need
RUN apt-get update && apt-get install -y libpng12-dev libjpeg-dev && rm -rf /var/lib/a
pt/lists/* \
    && docker-php-ext-configure gd --with-png-dir=/usr --with-jpeg-dir=/usr \
    && docker-php-ext-install gd
RUN docker-php-ext-install mysqli

VOLUME /var/www/html

ENV WORDPRESS_VERSION 4.2.2
```

Storage drivers are used to provide a single, collapsed view of these stacked layers. For operating system compatibility and performance sake, Docker supports several storage drivers. By default Docker will select a suitable driver for the platform you are working on.

Open Container Initiative

Although Docker pioneered container-based virtualization, today there are other solutions available on the market.

Although initially an avid supporter of Docker, CoreOS criticized the Docker container format for being overly complex and lacking security. Having seen no change, CoreOS decided to release its own container solution. CoreOS boldly claimed their solution had better security while maintaining full compatibility with Docker.

The competing container image formats threatened to fracture the entire community. Thankfully though, common sense prevailed and the community (including Docker and CoreOS) rallied together to form a common standard for container image formats to maintain interoperability across different container-based virtualization solutions. The Open Container Initiative (OCI) was started by the Linux Foundation in June 2015 to provide governance over a common container image format. The initial container standard is largely based on the Docker 2.2 image format, which had already included many features and improvements from CoreOS.

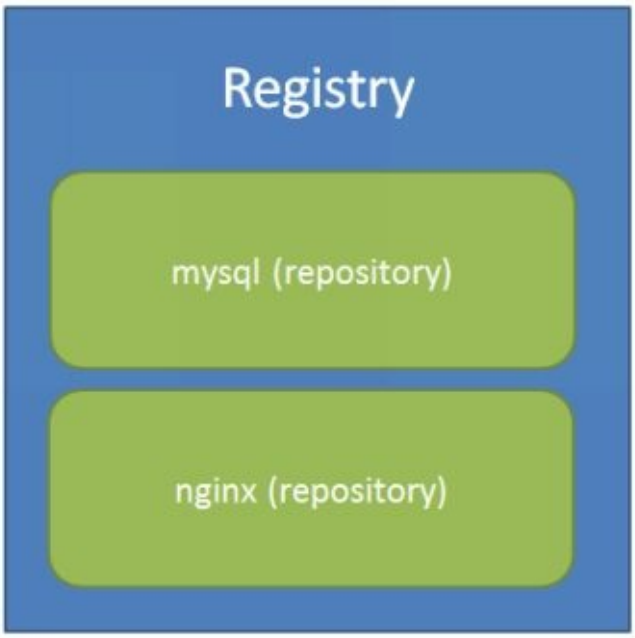
Various partners have formally committed to the OCI effort including, Google AT&T, Oracle,

SUSE, Twitter, Verizon. Amazon Web Services, Cisco, CoreOS, Docker, Fujitsu Limited, Goldman Sachs, HP, Huawei, IBM and Intel - just to name a few!

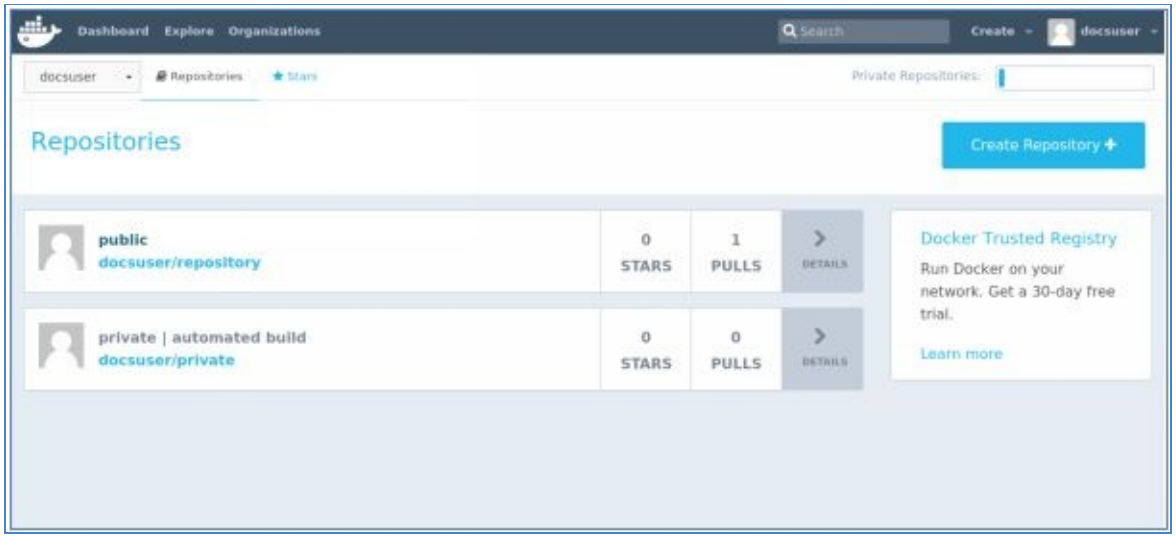
At the time of writing the OCI is set to soon release version 1.0 of the proposed container format specification.

Registries

Dockers default method for distributing images is known as a *registry*; a service responsible for hosting and distributing images. Images are stored within the registry in a centralised store known as a *repository*. Each registry may house multiple *repositories*, each isolated from one another. Registries can be hosted in the cloud or on-premises, accessible publically or privately.



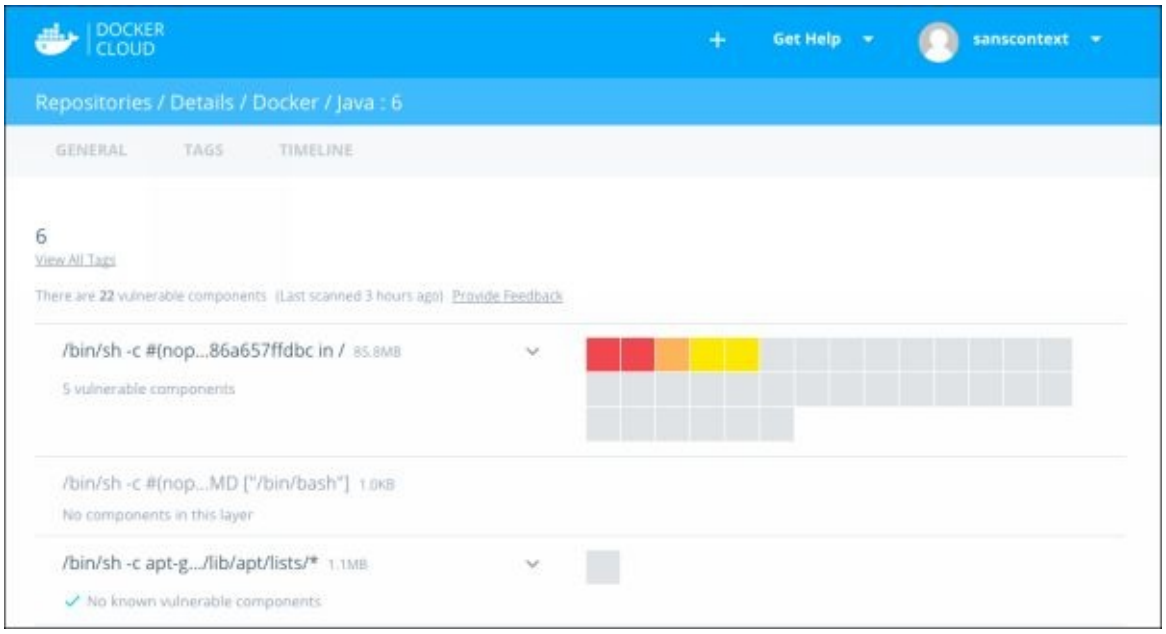
There are a number of available registries to choose from, including Dockers own Docker Hub and Docker Store. Although somewhat confusing, *Docker Hub* is primarily community focussed, while *Docker Store* is aimed at enterprise.



Other third party registries include:

- Google Container Registry
- Amazon EC2 Container Registry
- Microsoft Azure Container Registry
- Security vulnerabilities

An analysis of Docker Hub images has revealed that over **30% of official images** contained known security vulnerabilities, and over **40% for community uploaded images**. For this reason, Docker has implemented an image scanning solution to identify unpatched security vulnerabilities. Just like ordinary physical or virtual hosts, your base images and containers should be routinely patched. Generally you should patch your base images and re-create your containers, as opposed to patching your containers.


DOCKER CLOUD
Get Help
sanscontext
Repositories / Details / Docker / Java : 6
GENERAL
TAGS
TIMELINE
6
View All Tags
There are 22 vulnerable components (Last scanned 3 hours ago) Provide Feedback
/bin/sh -c #(nop...86a657ffdbc in / 85.8MB
5 vulnerable components
/bin/sh -c #(nop...MD ["/bin/bash"] 1.0KB
No components in this layer
/bin/sh -c apt-g.../lib/apt/lists/* 1.1MB
No known vulnerable components

Chapter 5

Docker History

In 2010, a small startup known as dotCloud were struggling to find their market fit for their PaaS solutions. Underpinning their core PaaS offering was a utility that the team had developed internally to simplify the power of *lxc*, enabling developers and operations staff to create and manage containers.

In March 2013, after seeing limited success with their PaaS product, founders Solomon Hykes and Sebastien Pahl decided to open source the utility they had developed, and Docker was born! The initial release consisted of a wrapper written around Linux lxc container technology, and was just 600 lines long!

Almost immediately, the industry began to take notice. Within 1 month of release, Docker claims that over 10,000 developers had already started using the platform.

In October 2013, dotCloud was later renamed to *Docker Inc*. Over the coming year the team released many new features for the product, most notably, in March 2014 Docker released version 0.9; switching to the libcontainer execution driver; all prior releases had relied on Linux lxc container technology. The primary reason for this move was for operating system compatibility. While lxc worked well it was heavily tied to the Linux platform. This essentially meant that the Docker Engine would not operate on other platforms such as Windows. For this reason, Docker developed the open source *libcontainer* library; providing an abstraction layer over the host operating system to remove dependencies on the underlying host platform.

In April 2015 Docker secures $95M in Series D funding, enabling the platform to grow to its full potential. The win reaffirmed the tremendous interest in the Docker platform and container technologies.

Docker announced that it has surpassed 1 billion downloads in Nov of 2015. A staggering milestone for the company! This staggering growth has only continued; as of June 2016 there has been more than 4.1 billion downloads of Docker Engine, a tremendous achievement in the span of a few short years!


32,000+
GitHub Stars
4B+
Docker Container Downloads
450,000+
Dockerized Apps In Docker Hub
250+
Meetup Groups In 70+ Countries
2900+
Community Contributors
95,000+
Third Party Projects Using Docker

Chapter 6

Moving to Containers

Excited about Docker and what it can provide for you?

Docker provides a huge value proposition for organisations looking to streamline their existing DevOps processes and enable massive scalability.

Before jumping into the deep end of the pool, there's a few things should consider:

Acceptance

The first step to successfully transitioning to Docker is to accept that change is inevitable - container-based virtualisation could be coming to your organization. Yes you may not have any plans right now to migrate to containers, but no one can predict the future so it's best to start preparing NOW!

Communication is Key

If you decide to move to containers, make it official. Make sure all your teams (including your developers and operations teams) are aware of the upcoming change, and are all on the same page.

Make sure your teams are involved every step of the way throughout the entire process to make them feel involved.

Planning

If developing new applications consider the following:

- Follow a microservices architecture to promote scalability and minimise effort required to migrate. Although Docker is capable of hosting stateful applications, it particularly excels with stateless applications.
- Platform agnostic: Your application should make no assumption about the platform it is running on. This mindset will encourage portability of your containers.

- Use common light-weight communication protocols such as HTTP endpoints using a REST API. Applications communicate across containers using the network stack as opposed to operating system specific communication buses.

Slow and Steady Wins the Race

It's not uncommon for organizations to fail to containerize their IT systems because they've bitten off more than they can chew, leaving a bad taste in the mouths of senior management.

Unless you have experienced operations staff familiar with your applications. Diving head first into Docker is not recommended, there's no rush!

There are a few things you need to consider, including upskilling your teams, new tooling, refining your deployment process, considerations for system patching etc.

Docker Engine Builds

Docker offers 3 different release builds. Choose wisely depending on your needs.

- **Experimental** – Nightly, bleeding edge builds for those that like to live on the edge.
- **Stable** – A new feature release typically every 2 months. Bug fixes are still routinely released as required.
- **Commercial**- Paid support offering for commercial customers – known as the *Commercially Supported (CS) Docker Engine*. Features are generally pushed out less frequently, typically every 6 months. Bug fixes are still routinely released as required.

It is recommended that enterprise clients choose the CS engine for SLA-based, guaranteed support. For everyone else, the stable build is recommended.

Security

Docker isn't perfect! It is theoretically possible that your host operating system could become compromised through unpatched exploits. Bugs such as kernel level exploit CVE-2015–8660 have previously been discovered in the overlay filesystem used by Docker. Docker shouldn't be used in a high security environment.

As always, make sure you keep up to date with the latest security patches.

Chapter 8

The Big Picture

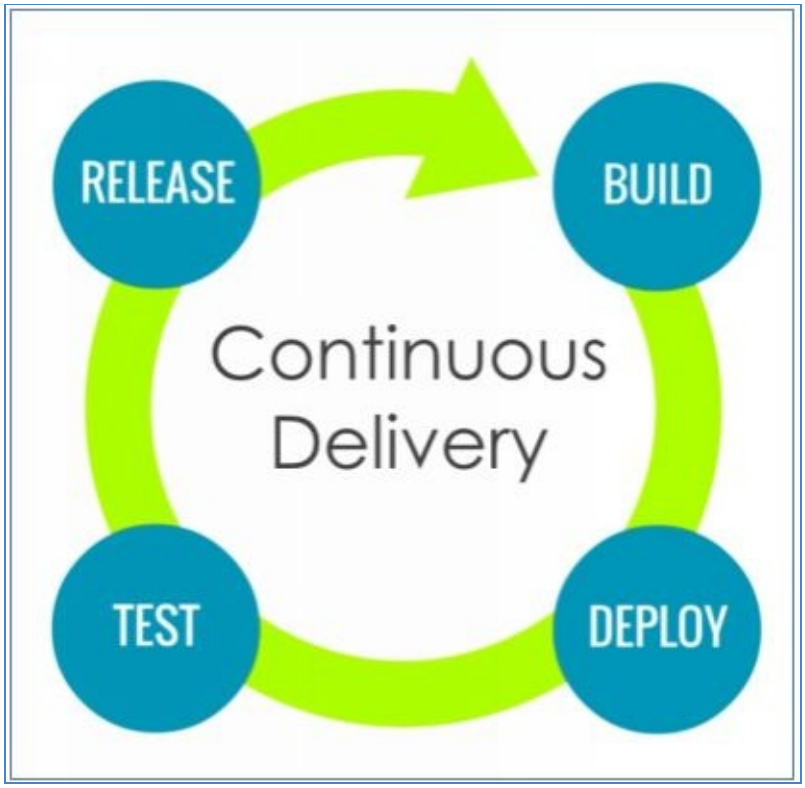
Docker isn't just about containers, it's provides an entire platform to streamline packaging, deployment and hosting of your applications.

Applications are typically composed of many smaller pieces such as a load balancer, front end web interface, backend web services, database tiers etc. Traditional approaches to managing these moving parts do not scale particularly well. You don't really want to be hand stitching your application infrastructure, piece by piece - introducing *container orchestration*.

Docker provides various orchestration tools to assist with this task:

- ***Docker Machine*** to manage your hosts
- ***Docker Compose*** to create and manage our app containers
- ***Docker Swarm*** to provide cluster management capabilities
- ***Docker Cloud*** (previously known as Tutum) provides a single interface to manage

The Docker platform greatly simplifies the process of establishing a continuous delivery workflow. In simple terms, developers committing new builds into source control will trigger automated tests and automated deployments thus enabling repeatable, reliable and quicker release cycles.


RELEASE
BUILD
Continuous
Delivery
TEST
DEPLOY

CHAPTER 9

ALTERNATIVES

While Docker is still considered the de facto standard for containers, today there are many other alternatives available on the market.

Some notable alternatives include,

- **RKT** (Rocket) – A highly popular containerization platform originally developed by CoreOS as a more security focused solution.
- **Ubunto** - Canonical (the makers of Ubunto) are set to soon release its own native containerisation platform built on top of LXD.
- **Windows Server 2016** - Microsoft have announced native containerization capabilities on the Windows Server platform.
- **Windows Azure Container Services (ACS)** – The cloud computing platform has recently released.

Conclusion

Thank You!

Firstly, I would personally like to thank you for purchasing this book. Hopefully this book has given you a fundamental understanding of Docker and containerization.

Containerisation is a fantastic new technology that is shaking up the industry.

I Need Your Help!

Did you enjoy this book? I would very much appreciate it if you could take the time to leave an Amazon review to help spread the word.

Sincerely,

John Symonds