The Definitive Guide To Docker Containers

Executive Summary

We are in a new technology age – software is dramatically changing. The era of off big packaged applications is shifting to an era defined by vibrant open source projects and lots of small, specialized applications. Driven first by the desire for a more agile application development model, the ripple effect has created a movement changing not only the technology but the people and process that build it.

Docker has emerged as a key enabling technology for this movement, from enabling DevOps methodologies to powering some of the most innovative websites/companies/applications in operation today. Popular amongst developers, Docker is gaining in popularity beyond open source devotees to large enterprises and even those beyond Linux.

This guide provides an introduction to the Docker technology, containerization, ecosystem and the resulting shift from monolithic to micro services architectures.

Intended for readers new to Docker, this paper will discuss the following topics:

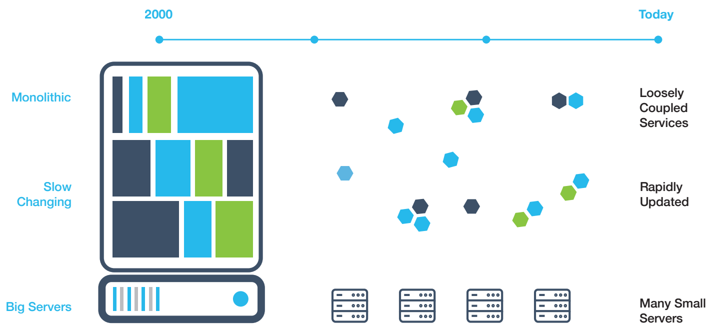
- Technology trends driving the change to distributed applications
- Overview of the Docker technology
- Impact to the application delivery ecosystem
- Key use cases and benefits

Distributed Applications Call for Containers

Many enterprises are moving away from monolithic applications to distributed applications. Fueled by a desire to innovate faster, organizations are moving away from waterfall style development practices traditionally linked to monolithic applications in favor of distributed applications and DevOps methodologies. With traditional applications, each additional line of code introduced grew the QA cycles exponentially and ultimately slowed innovation in favor of testing and bug fixing. The promise of distributed applications is that a collection of micro services can be developed and independently, frequently and freely.

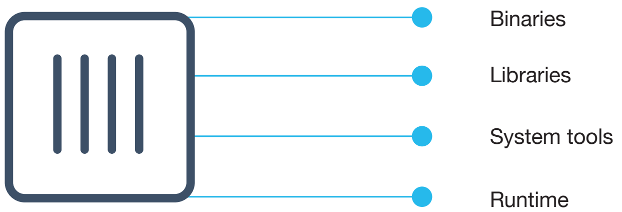
What is a distributed or micro services application? They fundamentally look and behave much differently than their predecessors. Instead of having a single application server containing an ever-expanding code base, that "application" is now a collection of loosely coupled smaller applications otherwise knows as "micro services". Instead of being written in a single language, each service can now be written in the best language stack and version for the capability it is intended to provide the end user. In an era of hybrid cloud, these services want to be distributed – running wherever is optimal from a performance and cost standpoint for that workload.

The idea behind micro services is that if one service fails, then the whole application does not have to fail and the developer can simply fix it, without affecting the other services that make up the application. Modern websites like Gilt, LinkedIn, Facebook and many others all use this type of architecture. Gilt, an online flash sale retailer has an ecommerce site comprised of roughly 400 applications. The code that serves up the pictures on the product page and the search functionality are actually separate applications. In the case of LinkedIn, if the "upload" capability fails, the entire LinkedIN application doesn't go down with it. The user simply gets the mes
sage "LinkedIN Upload capability is temporarily unavailable" instead of having the entire application freeze or crash.



This approach creates a new paradigm for building better software but is not also raises new challenges. With hundreds small applications connected together, new complexities are introduced with multiplicity of language stacks, versions, dependencies and hardware compatibility. Maintaining order becomes a challenge as a new matrix headache emerges.

Enter Docker, a platform for packaging up the application and all of its dependencies into a standardized unit of software – the Docker container.



Docker Basics

Before we dive into how the different components of the Docker technology, architecture and how they work, below are the core terminology and their definitions.

Dockerfile: This tells the image builder (i.e Jenkins) what the image should look like.

Image: The basis of a Docker container at rest. These artifacts are stored and managed in a registry. Once instantiated via a Docker run command a container is created.

Container: The standard unit in which the application service resides. At run, the image is turned into a container.

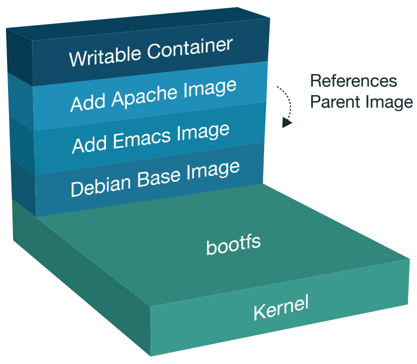
Docker Engine: Installed on physical, virtual or cloud hosts, this lightweight runtime is what pulls images, creates and runs containers.

Registry: A service where Docker images are stored, managed and distributed.

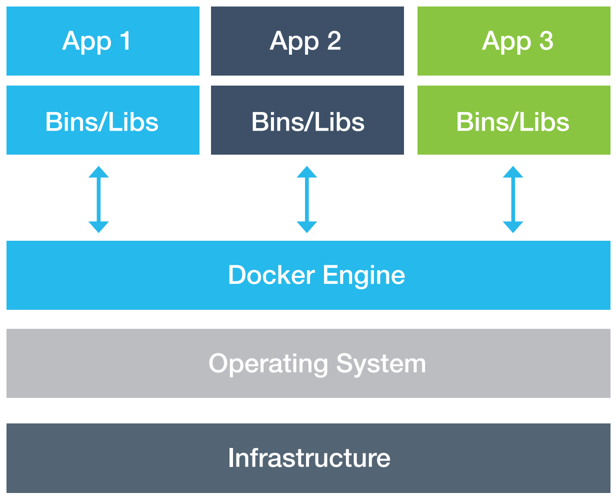
Understanding Docker Containers

A Docker container wraps up a piece of software in a complete filesystem that contains everything it needs to run: code, runtime, system tools, system libraries – anything you can install on a server. By encapsulating and isolating everything in a container, this guarantees that the container will always run the same, regardless of the environment it is running in.

Docker containers are built from Docker images, which use union filesystems and share image layers which further attribute to their lightweight nature.



Containers are created from an image with the "docker run" command. Once initiated, the Docker Engine spins up that container from the defined image (it can also spin them down). Every command is executed to the Docker Engine including creating new containers, scaling existing containers, stopping, removing and much more.



Docker containers make distributed applications:

Composable: Lightweight containers start instantly and use minimal resources because they share the kernel of the underlying operating system

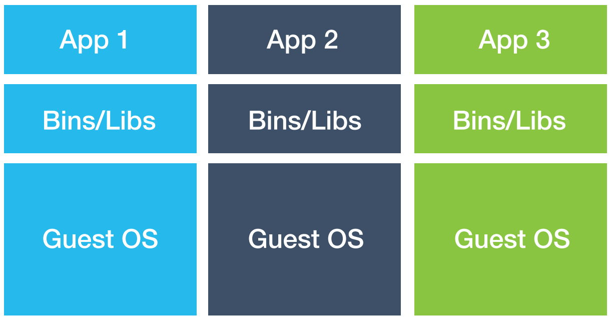
Dynamic: Quick to spin up and easy to scale and change make containers ideal for agile teams

Portable: Able to run on any physical, virtual or cloud environment running Docker.

Containers can uniquely turn very diverse set of application services into standardized software units. From the outside the containers all look the same which regardless of the binaries, libraries and code inside the container. This makes it easy to build tooling around container to move them along every stage of the application lifecycle. You can create simple single container or complex multi-container applications.

What about virtual machines?

But wait, what about virtual machines (VMs)? For years virtual machines were seen as the status quo for packaging and running applications. Containers and VMs seem similar, but containers utilize a different architectural approach. VMs require a full operating system (OS) inside of each VM in order to run the service inside of it. This allows for many OS's to run on the same physical server, which is beneficial for some use cases. On the other hand, containers share the OS kernel on the host thus being able to achieve higher levels of density per host but require the same OS.



VMs and containers are designed for different types of applications but are complimentary and have a natural intersection. Virtual machines are popular places to deploy and run Docker containers. Docker is able to run on physical and cloud infrastructure while uniquely provides portability for your applications across those environments. Additionally a single physical server could host a number of virtual machines with different OS's and each running their respective containers.

Docker Tooling and the Ecosystem

The beauty of containers is that they are a standard, lightweight, and extremely portable – to run in any environment on any infrastructure with sub second boot up times. Combined with open APIs and a pluggable architecture, Docker's unique characteristics have spawned a vibrant and vast ecosystem across the application delivery lifecycle.

Docker APIs are critical to a seamless Docker experience. As the Docker platform has grown beyond running a single container on a single Engine – the capabilities have expanded to automated provisioning (Machine), host clustering and container scheduling (Swarm), multi-container applications (Compose), image registries (Hub and Trusted Registry), security, networking, volumes to products that deploy and manage the Docker application environment (Universal Control Plane, Tutum) to complete the application lifecycle. These APIs enable the ecosystem to build value added tooling and services into the Docker platform.

Pluggable architecture uniquely gives the ability to maintain the Docker user experience with the flexibility to change out the underlying provider for technologies like; networking, storage, clustering, service discovery. Docker supports and maintains the plugin to the Docker Engine and third party providers build to the plugin API. This allows the same Docker commands to execute actions against a cluster, network or volume and the plugin translates the actions to the underlying provider.

Docker containers introduce an "application first" architecture, allowing teams to focus their time on building and shipping code through standardized units to package their software. Docker abstracts away the underlying infrastructure and compute, thus removing away the issues of "works on my machine" which plagues many software engineering, QA and IT Ops teams. This model also allows developers to granularly define requirements for their application at the container level like networking, volumes, resources, and more. Developers can develop, test and deploy applications, while IT Ops teams can provision, manage and secure infrastructure and applications across any environment.

Docker Use Cases

Today, the use of Docker tools within the enterprise has flourished. And of the enterprises using Docker tools, 40% are running Docker containers within their production environments, solidifying the tool as not just an awesome tool developing applications and making it easy to test them, but also for managing and deploying dockerized distributed applications in production. Teams have pulled over 1.3 billion images (and counting) from Docker Hub alone, using them as the basis for building their applications. Docker gives enterprises the agility, portability and control that their developers and operations teams require. Developers can create applications quickly, define the needs of each container, and can easily move containers from development and test environments. IT operations teams can keep up with the changes in business needs, and also control, manage and secure their environments, allowing them to meet the SLAs their customers expect. Here are a few use cases for Docker containers within the enterprise.

Continuous Integration/Continuous Delivery

ING, one of the top 10 largest financial firms in the world, was struggling with their container deployment process. Prior to using the containers it took up to 9 months for their applications to get deployed. Today, using the Docker platform, they are able deploy 1,500 times per week.

"Speed is very important to us and Continuous Delivery is how we do it." Says Kolk, "In ING IT, we are all developers and we want to be better - meaning better code and more customer satisfaction and Docker is the part of our strategy to do all of this, especially faster."

– Henk Kolk, Chief Architect of ING Netherlands

Infrastructure Optimization and Responsiveness

GILT GROUPE Gilt Groupe, the online flash sale fashion vendor, was using monolithics applications within their environment. They needed to keep up with the massive demand from their 6 million customers They took a very interesting approach in that they didn't do a dramatic shift to distributed apps. What they instead did was containerize each of the 7 monolithic applications they had. They then managed these containers, and made updates to them as needed.

> *"I love Docker. It's simple for developers and it works for ops. There is a really positive emotional connection that Docker has created with our developer community."*
>
> - Gilt Groupe

Containers-As-A-Service

Booz Allen Booz Allen in partnership with the GSA, the largest federal buying organization, used Docker to transform their monolithic applications to in an agile DevOps environment, using a containers as a service (CaaS) methodology. Making the government's testing more agile.

At Docker our mission is to help you build, ship and run distributed applications anywhere. Containerizing your applications and moving to micro services is just one piece of that journey. The power of the Docker platform is to deliver a secure and managed application environment that gives you speed and freedom in application development and delivery.

For more information take a look at the following resources:

- Learn more about Docker www.docker.com/enterprise
- Docker Self-paced Training: https://training.docker.com/self-paced-training
- Contact our sales team: www.docker.com/contact

www.docker.com