eBook

Docker for the Virtualization Admin

Table of Contents

Containers are not VMs

Docker is one of the most successful open source projects in recent history, and organizations of all sizes are developing plans around how to containerize their applications. The first step in this journey is, of course, to understand what containers are, and what are their key benefits.

A natural response when first working with Docker containers is to try and frame them in terms of virtual machines. Oftentimes we hear people describe Docker containers as "lightweight VMs".

This is completely understandable, and many people have done the exact same thing when they first started working with Docker. It's easy to connect those dots as both technologies share some characteristics. Both are designed to provide an isolated environment in which to run an application. Additionally, in both cases that environment is represented as a binary artifact that can be moved between hosts. There may be other similarities, but these are the two biggest.

The key is that the underlying architecture is fundamentally different between the containers and virtual machines. The analogy we use here at Docker is comparing houses (virtual machines) to apartments (Docker containers).

Houses (the VMs) are fully self-contained and offer protection from unwanted guests. They also each possess their own infrastructure – plumbing, heating, electrical, etc. Furthermore, in the vast majority of cases houses are all going to have at a minimum a bedroom,

living area, bathroom, and kitchen. It's incredibly difficult to ever find a "studio house" – even if one buys the smallest house they can find, they may end up buying more than they need because that's just how houses are built.

Apartments (Docker containers) also offer protection from unwanted guests, but they are built around shared infrastructure. The apartment building (the server running the Docker daemon, otherwise known as a Docker host) offers shared plumbing, heating, electrical, etc. to each apartment. Additionally apartments are offered in several different sizes – from studio to multi-bedroom penthouse. You're only renting exactly what you need.

Docker containers share the underlying resources of the Docker host. Furthermore, developers build a Docker image that includes exactly what they need to run their application: starting with the basics and adding in only what is needed by the application.

Virtual machines are built in the opposite direction. They start with a full operating system and, depending on the application, developers may or may not be able to strip out unwanted components.

For a lot of people these concepts are easily grasped. However, even when someone understands the architectural differences between Docker containers and virtual machines, they will often still try and adapt their current thoughts and processes around VMs to containers.

> **"How do I backup a container?"**
>
> **"What's my patch management strategy for my running containers?"**
>
> **"Where does the application server run?"**

To many the light bulb moment comes when they realize that Docker is not a virtualization technology, it's an application delivery technology.

In a VM-centered world, the unit of abstraction is a monolithic VM that stores not only application code, but often the stateful data. A VM takes everything that used to sit on a physical server and just packs it into a single binary so it can be moved around. But it is still the same thing.

With Docker containers the abstraction is the application; or more accurately a service that helps to make up the application.

In a micro-services architecture, many small services (each represented as a single Docker container) comprise an application. Applications are now able to be deconstructed into much smaller components which fundamentally changes the way they are initially developed, and then managed in production.

So, how does a sysadmin backup a Docker container? They don't. The application data doesn't live in the container, it lives in a Docker volume that is shared between 1-N containers as defined by the application architecture. Sysadmins backup the data volume, and forget about the container. Optimally Docker containers are completely stateless and immutable.

Certainly patches will still be part of the sysadmin's world, but they aren't applied to running Docker containers. In reality if someone patched a running container, and then spun up new containers based on an unpatched image, serious chaos could ensue. Instead admins update their existing Docker image, stop their running containers, and start up new ones. Because a container can be spun up in a fraction off a second, these updates are done in exponentially more quickly than they are with virtual machines.

Application servers translates into a service run inside of a Docker container. Certainly there may be cases where microservices-based applications need to connect to a non-containerized service, but for the most part standalone servers where application code is executed give way to one or more containers that provide the same functionality with much less overhead (and much better horizontal scaling). ⚓

Containers and VMs Together

So if containers are not VMs, a logical question is: Can VMs and Docker containers coexist??

The answer is a resounding "**yes**."

At the most basic level VMs (in all their forms) are a great place for Docker hosts to run. Whether it's a vSphere VM or a Hyper-V VM or an AWS EC2 instance, all of them will serve equally well as a Docker host. Depending on what you need to do, a VM might be the best place to land those containers. But the great thing about Docker is that, it doesn't matter where you run containers – and it's totally up to you.

Another question that is often asked relates to whether or not Docker container-based services can interact with VM-based services. Again, the answer is absolutely yes. Running your application in a set of Docker containers doesn't preclude it from talking to the services running in a VM.

For instance, your application may need to interact with a database that resides in a virtual machine. Provided that the right networking is in place, your application can interact with that database seamlessly.

Another area where there can be synergy between VMs and Docker containers is in the area of capacity optimization. VMs gained early popularity because the enabled higher levels of server utilization. That's still true today. A virtualization host, for instance, can host VMs that may house Docker hosts, but may also host any number of traditional monolithic VMs. By mixing and matching Docker hosts with "traditional" VMs, sysadmins can be assured they are getting the maximum utilization out of their physical hardware.
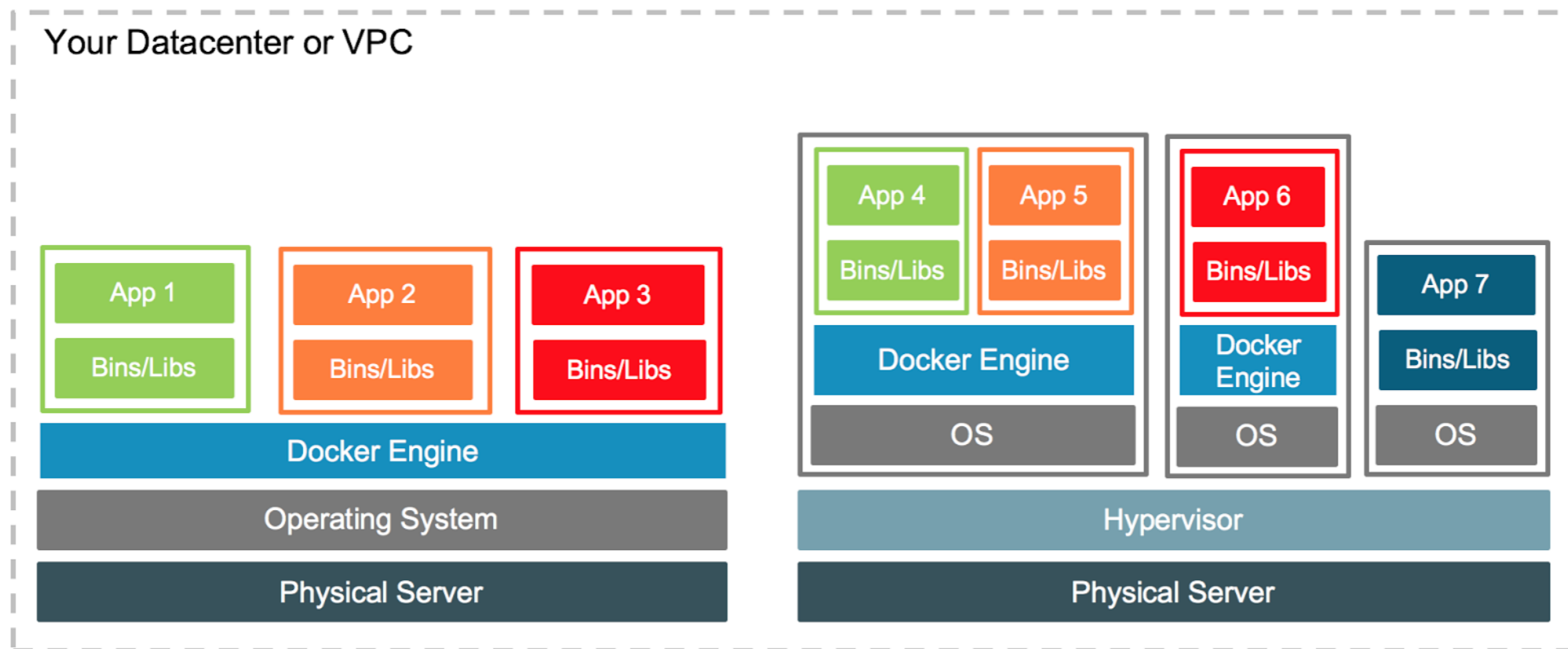
Docker embraces running Docker hosts on a wide variety of virtualization and cloud platforms. Docker Cloud and Docker Datacenter can easily manage Docker hosts regardless of where they run. And with Docker Machine you can provision new Docker hosts onto a wide variety of platforms including VMware vSphere, Microsoft Hyper-V, Azure, and AWS.

One of the most powerful things about Docker is the flexibility it affords IT organizations. The decision of where to run your

applications can be based 100% on what's right for your business. You're not locked into any single infrastructure, you can pick and choose and mix and match in whatever manner makes sense for you organization. Docker hosts on vSphere? Great. Azure? Sure. Physical servers? Absolutely. With Docker containers you get a this great combination of agility, portability, and control. ⚓

Modernize and secure the software supply chain with Docker.

Docker Datacenter delivers an integrated and robust platform for container orchestration and management that is flexible to integrate into existing systems and processes.

AGILITY

Deliver innovation at speed. Docker helps companies ship up to 13X more software.

PORTABILITY

Gain full stack portability of applications, networks, storage across any infrastructure.

CONTROL

Securely orchestrate and manage containers at scale at optimal costs.

Try Docker Datacenter free for 30 days
WWW.DOCKER.COM/ENTERPRISE

Physical or Virtual?

Virtual machines make great Docker hosts, but often companies wonder if containers would be better served running on bare metal physical servers.

And when they pose this question to Docker experts, the conversation goes something like this:

> **Docker Expert:** *It's not a question of "either / or" – that's the beauty of Docker. That choice is based solely on what's right for your application and business goals – physical or virtual, cloud or on premise. Mix and match as your application and business needs dictate (and change).*

> **User:** **But, surely you have a recommendation.**

> **Docker Expert:** *I'm going to give you the two word answer that nobody likes: "It depends."*

> **User:** **You're right, I don't like that answer.**

> **Docker Expert:** *I kind of figured you wouldn't, but it really is the right answer.*

There are tough questions in the world of tech, and the answer "It depends" can often be a way of avoiding them. But in the case of where to run your containerized applications it really is the best answer because no two applications are exactly the same, and no two companies have exactly the same business needs.

Any IT decision is based on a myriad of variables: Performance, scalability, reliability, security, existing systems, current skillsets, and cost (to name just a few). When someone sets out to decide how to deploy a Docker-based application in production all of these things need to be considered.

Docker delivers on the promise of allowing you to deploy your applications seamlessly regardless of the underlying infrastructure. Bare metal or VM. Datacenter or public cloud. Heck, deploy your application on bare metal in your data center and on VMs across multiple cloud providers if that's what is needed by your application or business.

The key here is that you're not locked into any one option. You can easily move your application from one infrastructure to another. There is essentially zero friction.

But that freedom also makes the process of deciding where to run those applications seem more difficult than it really is. The answer is going to be influenced what you're doing today, and what you might need to do in the future.

And, while there is no easy answer to this question, there are a number of things to consider when it comes time to make your decision.

The list here is probably far from complete, but hopefully it's enough to start a conversation and get the gears turning

Latency: Applications with a low tolerance for latency are going to do better on physical. This something we see quite a bit in financial services (trading applications are prime example).

Capacity: VMs made their bones by optimizing system load. If your containerized application doesn't consume all the capacity on a physical box, virtualization still offers a benefit here.

Mixed Workloads: Physical servers will run a single instance of an operating system. So, you if you wish to mix Windows and Linux containers on the same host, you'll need to use virtualization

Disaster Recovery: Again, like capacity optimizations, one of the great benefits of VMs are advanced capabilities around site recovery and high availability. While these capabilities may exist with physical hosts, the are a wider array of options with virtualization.

Existing Investments and Automation Frameworks: A lot of the organizations have already built a comprehensive set of tools around things like infrastructure provisioning. Leveraging this existing investment and expertise makes a lot of sense when introducing new elements.

Multitenancy: Some customers have workloads that can't share kernels. In this case VMs provide an extra layer of isolation compared to running containers on bare metal.

Resource Pools / Quotas: Many virtualization solutions have a broad feature set to control how virtual machines use resources. Docker provides the concept of resource constraints, but for bare metal you're kind of on your own.

Automation/APIs: Very few people in an organization typically have the ability to provision bare metal from an API. If the goal is automation you'll want an API, and that will likely rule out bare metal.

Licensing Costs: Running directly on bare metal can reduce costs as you won't need to purchase hypervisor licenses. And, of course, you may not even need to pay anything for the OS that hosts your containers.

In the end, there is something really powerful about being able to make a decision on where to run your application solely based on the technical merits of the platform AND being able to easily adjust that decision if new information comes to light.

In the end the question shouldn't be "bare metal OR virtual" – the question is which infrastructure makes the most sense for my application needs and business goals. So mix and match to create the right answer today, and know with Docker you can quickly and easily respond to any changes in the future. ⚓

Getting Started

A move to Docker has to start somewhere. Admins are being asked simultaneously to maintain existing legacy applications as well as roll out new ones. With Docker now in their technology toolbox, they often end up asking themselves where these applications should be run: in a VM or in a container.

We already know that containers and VMs can coexist, so there is not going to be a single answer to this question. As with every step in this journey, admins need to consider a series of different factors. With that context here are three scenarios to consider when deciding where to deploy your application.

1) **If you're starting from scratch on a new application (or rewriting an existing application from the ground up), and you're committed to writing it around a microservices-based architecture then containers are a no brainer.**

In many cases, companies will leave their existing monolithic applications in place, while they develop the next version using Docker containers and microservices

By leveraging Docker, companies can accelerate application development and delivery efforts, while creating code that can be run across almost any infrastructure without modification.

2) **You are committed to developing software based on microservices, but rather than wait until an application is completely rewritten, you want to begin gaining benefits of Docker immediately.**

In this scenario, companies will "lift and shift" an existing application from a VM into a Docker container.

With the monolithic application running in a container, the development teams can start breaking it down piece by piece. They can move some functions out of the monolith, and begin deploying them as loosely coupled services in Docker containers.

The new containers can interact with older, legacy applications (regardless of where they are running) as necessary, and over time the entire application is deconstructed, and deployed as a series of portable and scalable services inside Docker containers.
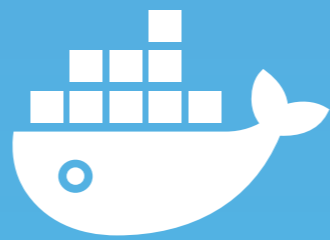
3) There are cases much like the second case, where companies want some benefits that Docker offers, and they move monolithic applications from VMs to containers with no intention of ever rewriting them.

Typically these customers are interested in the portability aspect that Docker containers offer out of the box. Imagine if your CIO came to you and said "Those 1,000 VMs we got running in the data center, I want those workloads up in the cloud by the end of next week." That's a daunting task even for the most hardcore VM ninja. There just isn't good portability from the data center to the cloud, especially if you want to change vendors. Imagine you have vSphere in the datacenter and the cloud is Azure — VM converters be what they may.

However, with Docker containers, this becomes a pretty pedestrian effort. Docker containers are inherently portable and can run in a VM or in the cloud unmodified, the containers are portable from VM to VM to bare metal without a lot of heavy lifting to facilitate the transition.

If any of these scenarios resonate with you, then you've probably got a good case to start trying Docker. ⚓

To learn more and get started today visit
www.docker.com/enterprise