Docker for the Virtualization Admin

Table of Contents

Containers are not VMs

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

A natural response when first working with Docker containers is to try and frame them in terms of virtual machines. They are oftentimes described as "lightweight VMs" and it's easy to connect those dots as both technologies share some characteristics. Two of the biggest similarities are that each is designed to provide an isolated environment in which to run an application, and that environment is represented as a binary artifact that can be moved between hosts. The key distinction is the underlying architectures of containers and virtual machines. While containers and virtual machines have similar resource isolation and allocation benefits, they function differently because containers virtualize the operating system instead of hardware. This lightweight form of encapsulation has the effect of making containers both more portable and more efficient than VMs. An excellent analogy to compare the two technologies is to consider 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, most houses 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. You also cannot just abandon a house and move when you are done with it. Selling and moving can be time-consuming, tedious, and expensive.

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 can rent exactly what you need. When you are done with the apartment you simply take your belongings and find a new one.

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.

To many the light bulb moment comes when they realize that Docker is not just virtualization technology that operates at a different layer, it's an application delivery and collaboration technology.

In a VM-centered world, the unit of abstraction is a monolithic VM that stores not only application code and dependencies, but also the entire operating system, and often the stateful data for the application as well. 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. Functionally, however, a VM is still treated like a server, including traditional backup & disaster recovery planning, operating system and application patching, and all the other traditional server tasks.

With Docker containers the abstraction is the application; or, in the move towards microservices, a single service that makes up an application. In a microservices 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. In the apartment analogy used earlier, each room and maybe even each appliance might be a microservice. You are now free to completely customize the size and shape of your apartment and individually monitor how each is doing. Maybe you only need that second bedroom when your kids or friends come to visit - with a microservice apartment you just add the new room when you need it and take it away when your guests leave.

How do I backup a container?

So, how does an administrator 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 one or more containers as defined by the application architecture. The volumes are backed by enterprise and cloud storage, so system admins backup the data volume, and forget about the container. Optimally, Docker containers are completely stateless and immutable.

What's my patch management strategy?

Certainly patches will still be part of the admin's world, but they aren't applied to running Docker containers. In production, if someone patched a live running container, and then spun up new containers based on the original 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 of a second, these updates are done much faster than they are with virtual machines. No need to reboot an entire OS - just stop the old version of the containerized service and start it again from the newer image. In our apartment model, we no longer wait for a repair technician to come and check out the oven if it stops working, we take out the old oven and replace it with a new one almost instantly. If we had a turkey in the oven, representing our stateful data, we just "detach" the turkey from the old oven and pop it back in the new one.

Where does the application server run?

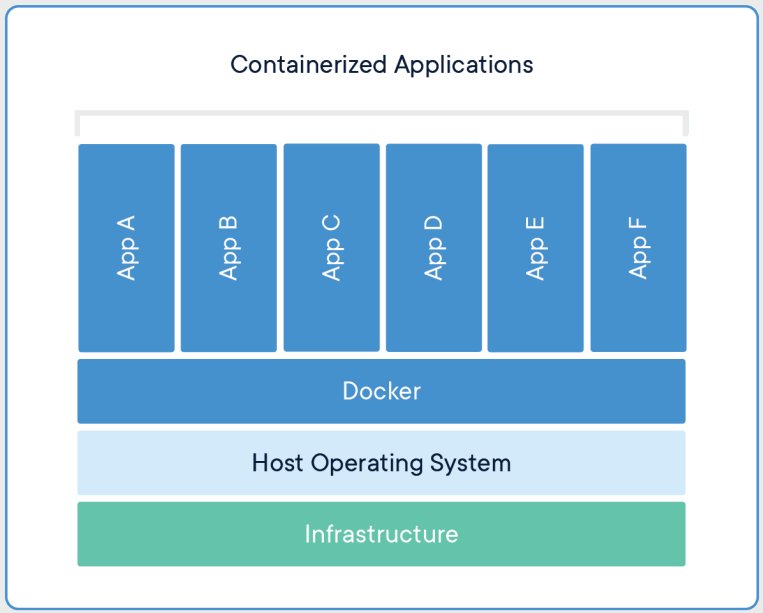
Application servers translate into a service run inside of a Docker container. There may be cases where microservices-based applications need to connect to a non-containerized service. For the most part, standalone servers where application code is executed, give way to one or more containers that provide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

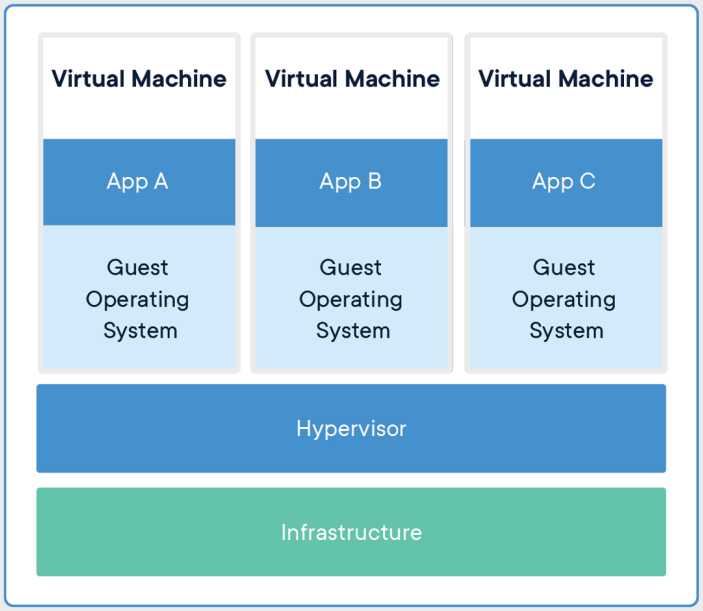
Containers and VMs Together

At the most basic level VMs in all their forms are a great place for Docker hosts to run. Whether you use vSphere, Hyper-V, KVM 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, or any other kind of server, for that matter.

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

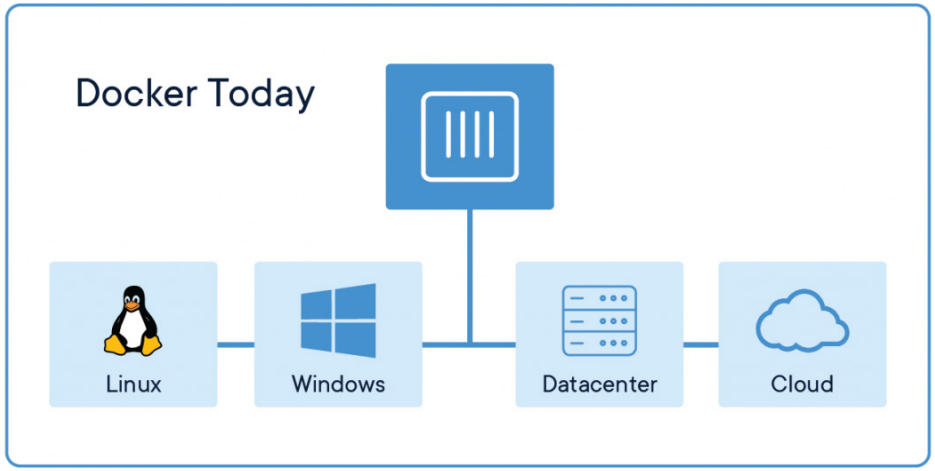
Another area where there can be synergy between VMs and Docker containers is in the area of capacity optimization. VMs gained early popularity because they enabled higher levels of server utilization. That's still true today. A virtualization host, for instance, can host VMs that may, in turn house Docker Engines, 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 Enterprise can easily manage Docker hosts regardless of where they run. You can provision new Docker hosts onto a wide variety of platforms including VMware vSphere, Microsoft Hyper-V, Azure, and AWS—even mainframes.

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 operators can securely run a container anywhere, from hybrid cloud to edge, all without lock-in.



Physical or Virtual?

Virtual machines make great Docker hosts, but oftentimes companies wonder if they would be better served running their containers directly on physical servers.

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-premises. 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 skill sets, 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, data center or public cloud. You can 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 by 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 far from complete, but 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 a prime example.

Capacity: VMs made their bones by optimizing system load. If your containerized applications don't consume all the capacity on a physical box, virtualization still offers a benefit. Plus, with VMs you can carve up your physical resources and dedicate them to groups of users; think of a sandbox environment for a developer team, as an example. The users feel like they have complete control of their own resources, and as an admin you get to set the boundaries.

Mixed Operating Systems: Physical servers will run a single instance of an operating system and services need to run on their original kernel architecture after they are containerized. 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 optimization, one of the great benefits of VMs are advanced capabilities around site recovery and high availability. While these capabilities may exist with physical hosts, there are a wider array of options with virtualization.

Existing Investments and Automation Frameworks: Many 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 or resources with other workloads. 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 and prioritize how virtual machines use shared resources. Docker provides the concept of resource constraints for a container, but if you are concerned about bad actors who might ignore these constraints VMs can add another boundary layer.

Automation/APIs: Very few people in an organization typically have the ability to provision bare metal from an API. If automation is one of the goals 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.

Get Started

A move to Docker has to start somewhere. Admins are being asked simultaneously to maintain and improve existing legacy applications as well as roll out new applications at a faster pace than ever. 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. **Greenfield Applications: New application development or rewriting an existing application from scratch.**

For organizations committed to modern DevOps practices, CI/CD and fast release cycles, containers are a no brainer. For new applications, this new application architecture allows organizations to incorporate microservices, in whole or in part from the start.

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

2. **Brownfield: Applications that are under active development but are based on legacy code bases.**

You can begin gaining the benefits of Docker immediately in this scenario. Entire applications can "lift and shift" 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.

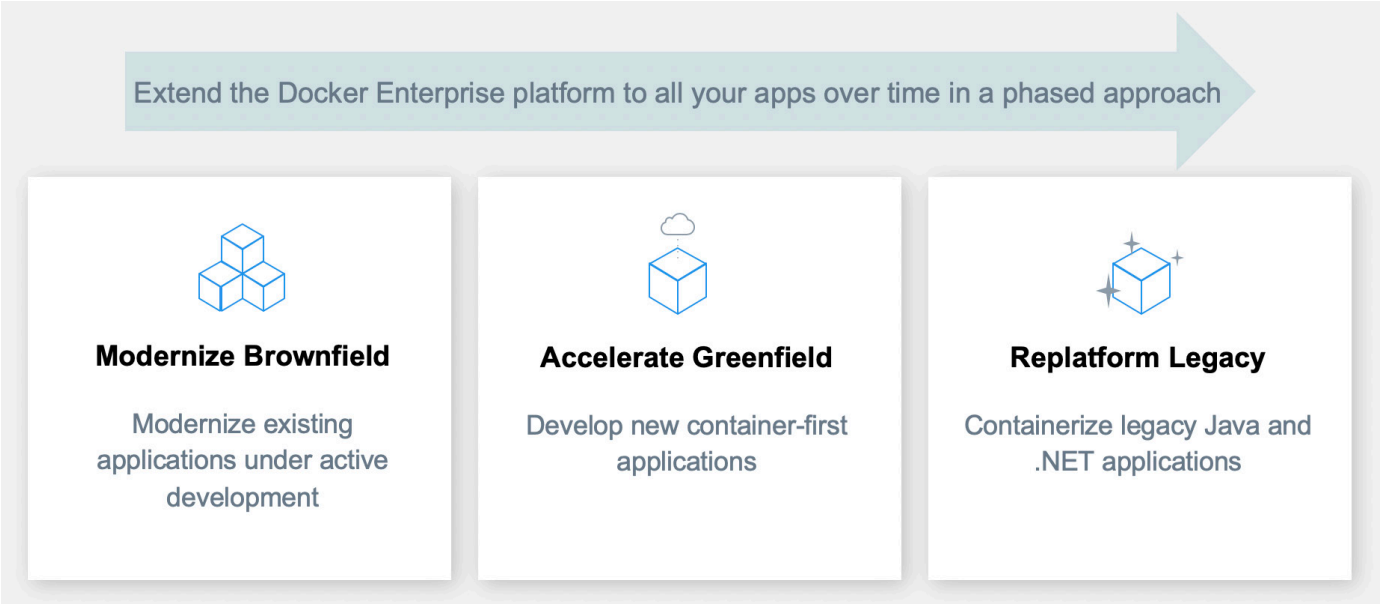
3. **Legacy: Move monolithic applications from VMs to containers with no intention of ever rewriting them.**

There are cases much like the second case, where companies want to extend the life of existing applications while making them more portable and easier to operate, but don't plan to touch the application code. Containerizing these legacy applications delivers significant cost savings for organizations through increased CPU utilization and server consolidation.

Imagine if your CIO came to you and said "Those 1,000 VMs we have 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. Plus, most of what you'd be transferring with VMs would be the base operating system in each VM - a complete waste of money and bandwidth. Then on top of it you have potential VM format conversions, manual recovery and restart planning and testing, and a new set of tools to use to manage everything in the cloud.

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 onsite VM to cloud VM to bare metal without heavy lifting to facilitate the transition. And in the world of Docker, each application has a Compose file or Kubernetes YAML to start the application, connect it to shared storage, and wire everything up, including security and even your management and monitoring tools.

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



About Docker

Docker Enterprise is the leading container platform for continuous, high-velocity innovation. Docker is the only independent container platform that enables developers to seamlessly build and share any application—from legacy to modern—and operators to securely run them anywhere—from hybrid cloud to the edge.

As the only independent container platform vendor, Docker enables customers to pursue the right strategy for their business and adapt to new technologies, without lock-in. For developers, this means using the best tools, languages and application stacks for each project. For IT, this means that they can pursue the right operational strategy for their business—any app on any OS, across any infrastructure. And with intrinsic end-to-edge security, the Docker Platform continuously ensure compliance and risk mitigation without slowing down innovation.

To learn more and get started today visit: https://dockr.ly/dockerenterpriseproducts

Contact Sales for more Information: https://dockr.ly/contactsales