CODESHIP

MANUEL WEISS
COFOUNDER OF CODESHIP

Why Containers and Docker are the Future

About the Author.

Manuel Weiss is cofounder of Codeship.
He founded the company in 2011 with the vision to help software development teams across the world become more efficient.

Codeship is a hosted Continuous Integration and Delivery Platform that focuses on speed, security and customizability.

Learn more about Codeship here.

Share this Codeship Guide

Why Containers and Docker are the Future.

When you have a look at the state of software development in 2015, two things jump out: Containerization and Docker.

In this ebook, I'll talk about why exactly containers are such a big deal and how Docker has helped revolutionize the way the software industry is using containers.

ABOUT THIS EBOOK

On the following pages we will compare three common tech stacks. The traditional (bare metal) stack, the virtual machine stack, and the container stack. You will learn about their advantages and disadvantages. Later on we will have a look at Docker and third party tools and services and how they help standardize the container workflow.

What Exactly Is a Container?

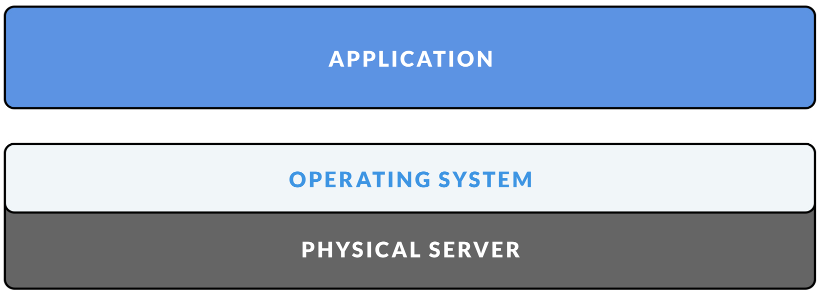
First, let's start off with a short history lesson and have a look at three different setups for providing a stack to run an application on. This will help us recognize what a container is and what makes it **so much more powerful** than other solutions. It will also show the many advantages of a container workflow.

Here are the three different setups we'll discuss:

- Traditional servers (bare metal)
- Virtual machines
- Containers

The Traditional Server Setup

The traditional server stack consists of a physical server that runs an operating system and your application.



This setup was the standard solution for many years. Its drawbacks eventually led to the advancements of the virtual machine stack and the container stack. Before that, a lot of companies, like Puppet Labs and Chef, had tried to work on solutions to tackle the drawbacks of the traditional server stack.

Advantages of the traditional server stack

There are a few good reasons why the traditional server stack still gets used.

Utilization of raw resources

One especially big bonus of using a traditional server stack is that your application can utilize a much higher amount of the available resources. This is due, in part, to the lack of overhead in running the application so close to the bare metal server — for example there is no hypervisor needed in this setup.

Isolation

Your application and data can be physically isolated from your other applications, as well as other users. By comparison, some container and virtual machine environments are shared.

Disadvantages of the traditional server stack

There are quite a few disadvantages to this setup though, which have prompted the evolution of other, more modern solutions.

- **Very slow deployment time**

  You actually have to rent a server and provision it. Also, making changes often requires a physical presence in a datacenter.

- **Expensive**

  Physical servers provide the best cost per resource, but they're somewhat inflexible. At scale, this cost can be absorbed, but with an application at low scale and with varying resource requirements, the cost per resource consumed becomes much higher.

- **Wasted resources**

  It's rare for a single application to need all the resources that a physical server provides. You're paying for the complete physical server, yet your application only needs 40 percent of its resources. Multi-tenancy (which is provided automatically by VMs and containers, but this is a manual process for the traditional stack) gives better value. Keep in mind that it also increases complexity.

Difficult to scale

Buying additional servers, setting them up, and getting them configured takes time. This process often depends on the responsiveness of the hosting provider used, as well as their set of supported operating systems.

Difficult to migrate

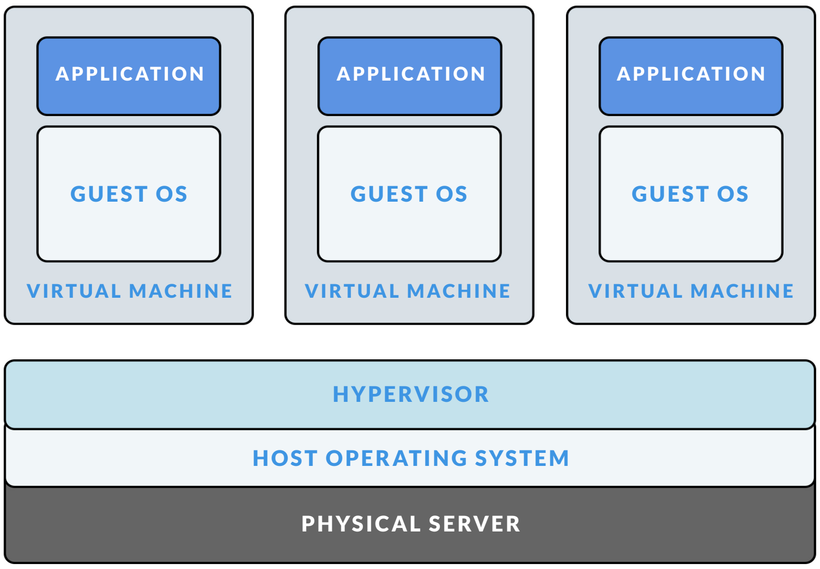
In a lot of cases, it's really hard to migrate from one server setup to another. Often no two servers are exactly the same, and there can be problems when migrating applications.

Complex configuration

Many tools exist to ease the preparation of a server for a specific application, however, it's still a difficult process. Maintaining a consistent environment across servers is very hard.

The Virtual Machine Setup

The VM stack consists of a physical server which runs an operating system and a hypervisor that manages your virtual machines, shared resources, and networking interface. Each Virtual Machine runs a Guest Operating System, an application or a set of applications.



Let's have a look at the advantages and disadvantages of this setup. You will see why the benefits of this setup were a big reason why the virtual machine stack has, in most cases, superseded the traditional server stack.

Advantages of the VM stack

There are many advantages to this setup when you compare it to the traditional stack.

Good use of resources

One physical server is divided into various virtual machines. The hypervisor can share resources based on priority, and VMs can make use of unused cycles that other VMs are not using. Each VM uses its own CPU, RAM, storage, and network allotment. Each VM also runs its own operating system.

Easy to scale

If you need additional applications running, you can always spin up more virtual machines, as long as your host server cluster still has resources left.

Easy to backup and migrate

It's very easy to migrate a virtual machine setup to different hardware. Most vendors provide simple mechanisms to backup, restore, and migrate stopped or running virtual machines, as well as automate failover support across the cluster.

- **Cost efficiency**

  Instead of monthly contracts, virtual machines can get billed by the hour, making responsive scaling to load possible.

- **Flexibility**

  VMs of different architectures can run alongside each other (for example, Windows + Linux).

Disadvantages of the virtual machine stack

The VM stack still has some drawbacks though, which led to the evolution of the container stack.

- **Resource allocation is problematic**

  It's hard to guarantee resources for a specific VM when all resources are shared.

- **Vendor lockin**

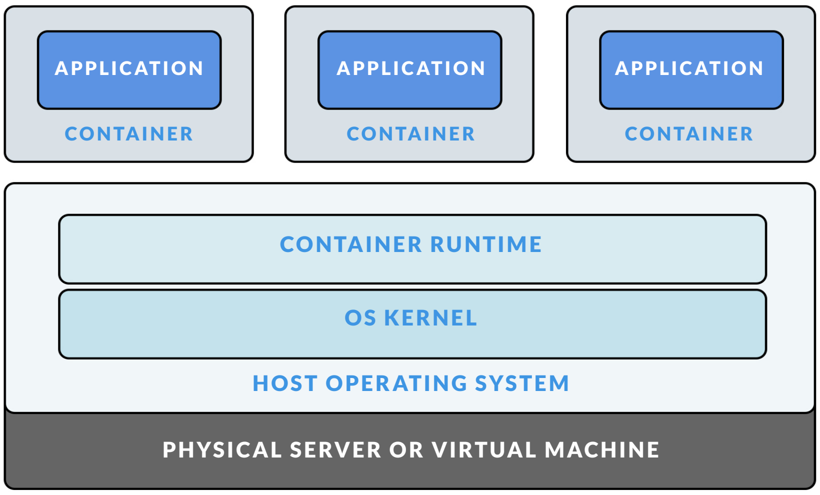
  Using one virtualization solution makes it hard to move to another. Some migration tools exist, but each stack has a different set of APIs and formats.

Complex configuration

Running an application inside a VM can still be troublesome. While the operating system can be easier to manage, preparing it for an application can still be a cumbersome process. Different instances of the same image can drift apart in terms of configuration and dependencies.

The Container Setup

The key difference between the container setup and the aforementioned solutions is that container-based virtualization **uses the kernel of the host OS to run multiple isolated guest instances**. These guest instances are called containers. The host can be either a physical server or a virtual machine.



By making use of the host operating system's kernel and not relying on a hypervisor to manage resources, the container stack is **far more lightweight than other solutions**. It also allows for completely isolated instances. Each container has its own root filesystem, processes, memory, devices, and network interface.

Advantages of the container stack

In addition to all the advantages the VM stack brings, the container stack also offers the following benefits.

Isolation

Each application runs in its own container instance. Because your code is in a container, it should run consistently across instances. Configuration worries can be reduced to a minimum by using containers. There's no need to worry about the end user's operating system if you ship the OS with the software.

Lightweight

By making use of the OS's kernel and not relying on a hypervisor, the container stack needs a lot fewer resources than any other setup. This allows for more containers in a machine than would be possible with VMs.

Resource effective

Containers use the host OS, which means there is no need for guest Operating Systems. This frees up a lot of resources. Many optimized operating systems exist, designed only to run containers.

Many simplified base container images targeted for certain application environments are being developed. This means that, with a containerized application, the aggregated overhead is minimized all the way down to the bare metal server.

Easy to migrate

Because each container is an isolated instance that doesn't hold a guest operating system, it's very easy to migrate to from one deployment to another. The container stack allows for great portability.

Security

Having each app run in an isolated instance also allows for great security. However, keep in mind that, depending on the container technology, there may be some holes in multi-tenanted container environments.

Low overhead

Due to the shared kernel, it's very cheap to start and stop containers.

- **Mirror production and development environment**

  Each app is running in an isolated container. These isolated containers can be shipped easily from your local machine to your hosting environment making the old "**But it worked on my machine!**" excuse a thing of the past.

- **Community**

  We'll talk about Docker in a minute, but it has to be mentioned that in recent times, container virtualization has evolved by leaps and bounds due to the help of many contributors from all over the world. Docker has especially helped with this; it standardized the container workflow with open-source tools and has built a vibrant community. Should you run into a problem, the chances are high that somebody else either has already encountered the problem and can provide help or that your problem is actively being worked on.

Disadvantages of the container stack

Some of the perceived advantages of a container setup could turn out to be disadvantages for your specific situation and might not be what you really need the most. Your final decision should always be about what your product and company needs.

Architecture

Containers share the kernel of the host OS, which means they're limited to the same architecture. Container images can be built for different architectures, but they're not portable across different architectures.

Resource heavy apps

Large, resource-heavy apps may be better suited to a traditional server setup where they have more direct access to the resources provided by the server.

Docker as a Standardized Container Platform

By comparing these three tech stacks, we can see why containers are such a big deal. But it has to be said that container virtualization is **not a new concept**. As a matter of fact, this technology has been known for more than 30 years. The concept was there, but recently Docker (and some of its competitors) have started building out standardized platforms and a community around containers. That's why we've heard so much about Docker lately.

On their homepage, Docker says it's "**a platform for developing, shipping, and running applications using container virtualization technology.**"

What makes this **platform** so great is that with Docker, you get more than just access to a set of high-quality core tools. You also get access to a community and a whole ecosystem of third-party products and services that help you from the very first steps of developing your application through every incremental deployment of it. Codeship, for example, is one of these third-party services.

The **Docker core platform** consists of a variety of products and services:

- Docker Engine (Docker Daemon)
- Docker Hub (Registry)
- Docker Machine (Host provisioning)
- Docker Swarm (Host clustering)
- Docker Compose (Container Orchestration)
- Docker Orca (Application management)
- Kitematic (GUI)

You can download ebooks and guides about each of these products in our Codeship Resources Library.

Why Containerization and the Docker Platform are the way forward

By comparing the container stack with its predecessors, we learned that **containerization is the fastest, most resource-effective, and most secure setup we know to date**. Containers are isolated instances that run your applications. These lightweight instances can be replaced, rebuilt, and moved around easily. This allows us to **mirror the production and development environment** and is a tremendous help in the continuous integration and delivery process. The advantages containers can provide are so compelling that they're definitely here to stay.

To learn more about containers and continuous delivery, check out our ebooks about Automating your Development Workflow with Docker and Continuous Integration and Delivery with Docker. I've added links to both for you at the end of this ebook. You can also find an ebook about Orchestrating your Containers with Docker Compose there.

Conclusion

Containerization has been here for a long time, but with the rise of Docker, containerization virtualization got standardized. The industry now benefits from a **standardized container workflow** and an **ecosystem** of helpful tools, services, and a vibrant community around it.

Lastly, keep in mind that while working with a container stack is certainly great, there are still a lot of scenarios where a traditional server stack or a VM stack might make more sense for you. In no way are these setups displaced by the container stack.

If you enjoyed this ebook you might like our others as well! In our Codeship Resources Library we provide guides, courses, and books about efficient software development, Continuous Integration and Delivery, Docker, the Docker Tool Chain, and much more.

CODESHIP

Share this Codeship Guide

Further Reading

DOCKER

- The Future is Containerized
- Container Operating Systems Comparison
- Building a Minimal Docker Container for Ruby Apps

CONTINUOUS DELIVERY

- Running a MEAN web application in Docker containers on AWS
- Running a Rails Development Environment in Docker
- Testing your Rails Application with Docker

Share this Codeship Guide

More Codeship Resources.

Share this Codeship Guide

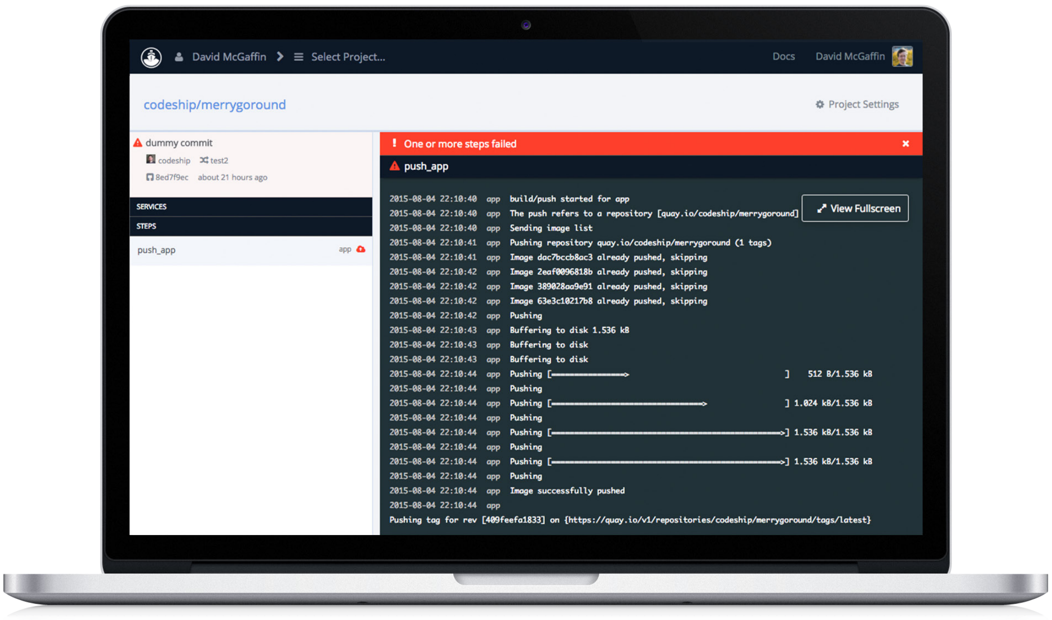
About Codeship.

Codeship is a hosted Continuous Integration and Delivery Platform that focuses on speed, security and customizability.

Connect your GitHub and Bitbucket projects and Codeship will test and deploy your apps at every push. Create **organizations**, teams and handle permissions to ensure smooth collaboration and **speed up your testing 10x** with Codeship's ParallelCI pipelines.

You can learn more about Codeship here.

https://codeship.com



```
2015-08-04 22:10:40  app  build/push started for app
2015-08-04 22:10:40  app  The push refers to a repository [quay.io/codeship/merrygoround]
2015-08-04 22:10:40  app  Sending image list
2015-08-04 22:10:41  app  Pushing repository quay.io/codeship/merrygoround (1 tags)
2015-08-04 22:10:41  app  Image dac7bccb8ac3 already pushed, skipping
2015-08-04 22:10:42  app  Image 2eaf0096818b already pushed, skipping
2015-08-04 22:10:42  app  Image 389028aa9e91 already pushed, skipping
2015-08-04 22:10:42  app  Image 63e3c10217b8 already pushed, skipping
2015-08-04 22:10:42  app  Pushing
2015-08-04 22:10:43  app  Buffering to disk 1.536 kB
2015-08-04 22:10:43  app  Buffering to disk
2015-08-04 22:10:43  app  Buffering to disk
2015-08-04 22:10:44  app  Pushing [================>                                  ]    512 B/1.536 kB
2015-08-04 22:10:44  app  Pushing
2015-08-04 22:10:44  app  Pushing [=================================>                 ] 1.024 kB/1.536 kB
2015-08-04 22:10:44  app  Pushing
2015-08-04 22:10:44  app  Pushing [==================================================>] 1.536 kB/1.536 kB
2015-08-04 22:10:44  app  Pushing
2015-08-04 22:10:44  app  Pushing [==================================================>] 1.536 kB/1.536 kB
2015-08-04 22:10:44  app  Pushing
2015-08-04 22:10:44  app  Image successfully pushed
2015-08-04 22:10:44  app
Pushing tag for rev [409feefa1833] on {https://quay.io/v1/repositories/codeship/merrygoround/tags/latest}
```