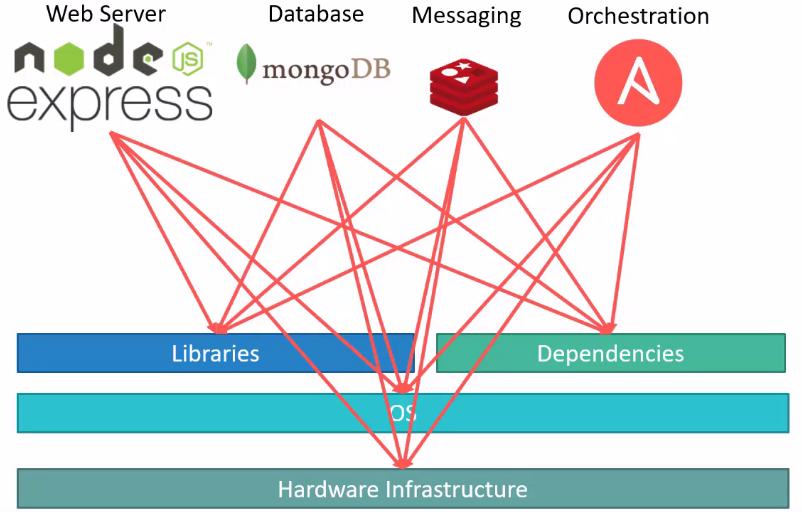
Kubernetes, also known as K8s, was built by Google based on their experience running containers in production. It is now an open-source project and is arguably one of the best and most popular container orchestration technologies out there. To understand Kubernetes, we must understand two things:

* Container + Orchestration

One of the most popular Container technologies is Docker. Let’s understand why we need Docker and what can it do for us?



Let’s assume you are building an application using the above technology stack, you may encounter the below issues:

1. Compatibility with the underlying operating system version.
2. Compatibility between the services and the libraries.
3. The architecture of the application may change over time.

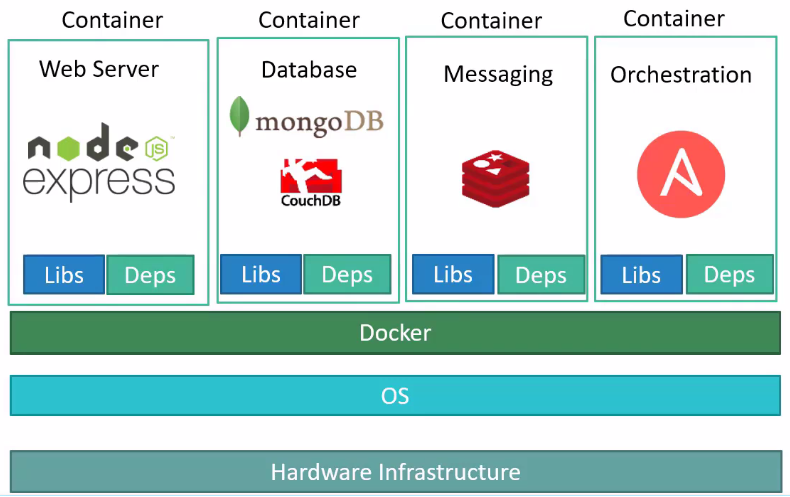
This compatibility matrix issue is usually referred to as The Matrix from Hell!

New developers may find it difficult to set up their environment. They have to follow a large set of instructions and run hundreds of commands to set up their environment varying from OS version, libraries etc.

We will have different development, QA, and production environments. And we couldn’t guarantee that the application we are building would run the same way in different environments. So, this will make your life difficult.

So, we need something that could help us with the compatibility issues. Something that will allow us to modify or change these components without affecting the other components and even modify the underlying operating system as required. That something is Docker.

With Docker, you can run each component in a separate container with its own libraries and its own dependencies, all on the same VM and the OS, but within separate environments or containers. We just have to build the Docker configuration once and all our developers can now get started with a simple Docker run command, irrespective of what the underlying operating system they run, all they needed to do was to make sure they had Docker installed on their systems.



What are Containers?

Containers are completely isolated environments, they can have their own processes or services, their own networking interfaces, their own mounts, just like virtual machines, except they all share the same operating system kernel. Containers are not new, they have existed for about ten years now and some of the different types of containers are LXC, LXD etc.

Docker utilizes LXC containers. Setting up these containers’ environment is hard as they are very low level and that is where Docker offers a high-level tool with several powerful functions, making it really easy for end users like us. To understand how Docker works, let us revisit some basic concepts of operating systems first.

If you look at operating systems like Ubuntu, Fedora, CentOS they all consist of two things, an OS kernel, and a set of software. The operating system kernel is responsible for interacting with the underlying hardware, while the OS kernel remains the same, which is Linux in this case it’s the software above it that makes these operating systems different.

The software may consist of a different user interface, drivers, compilers, file managers, developer tools etc. So, you have a common Linux kernel shared across all operating systems and some custom software that differentiates operating systems from each other. What does sharing the Kernel means?

Let’s say we have a system with an Ubuntu OS with Docker installed on it. Docker can run any flavor of OS on top of it as long as they’re all based on the same kernel, in this case Linux. If the underlying operating system is Ubuntu, Docker can run a container based on another distribution like Debian, Fedora, SUSE or CentOS.



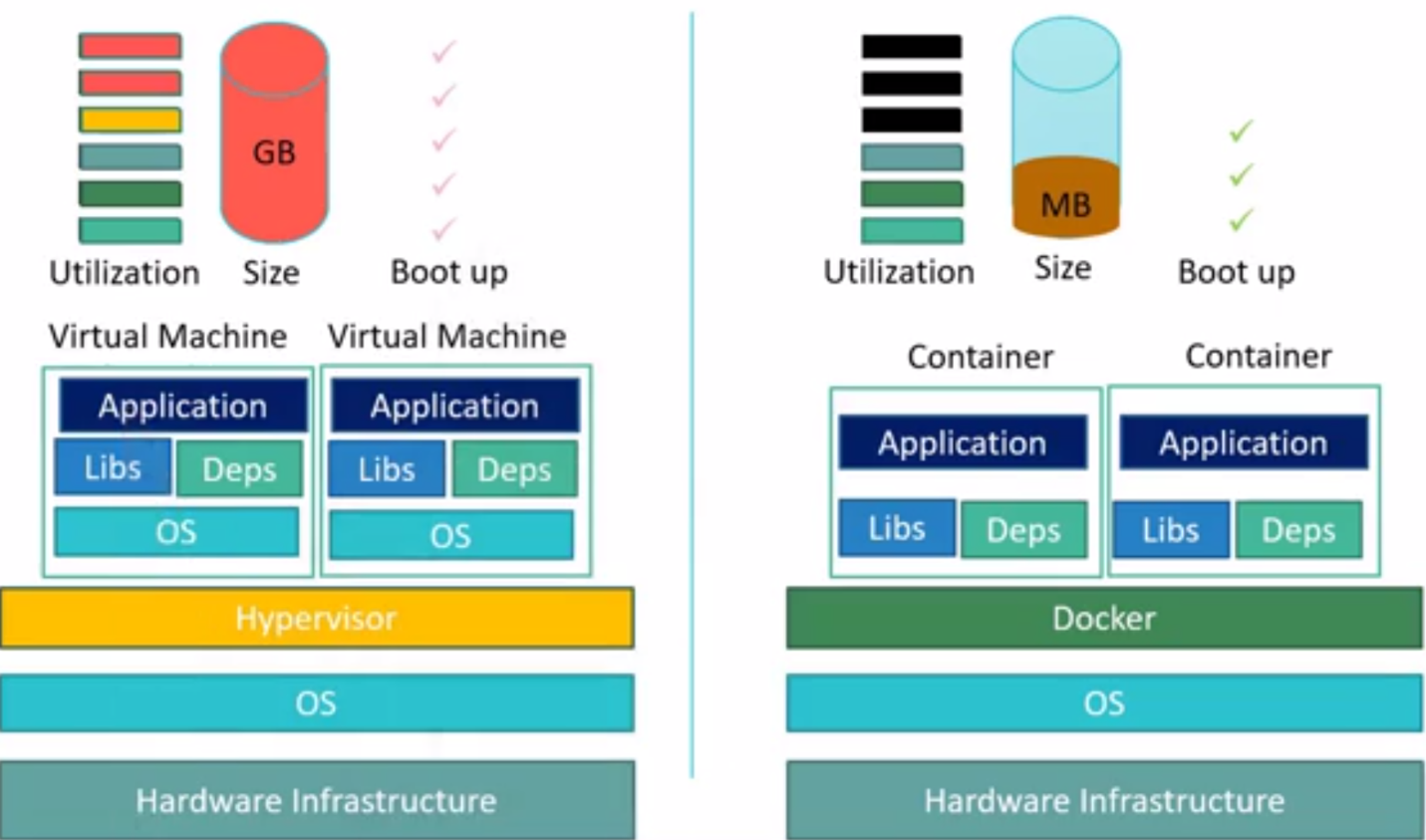
Each Docker container only has the additional software that makes them different and Docker utilizes the underlying Kernel of Docker host, which works with all the Operating Systems above. So, what is an OS that do not share the same Kernel as this?

Windows, so you won’t be able to run a Windows based container on a Docker host with Linux OS on it. For that you would require a Docker on a Windows Server. So, is it a disadvantage then, not being able to run another Kernel on the OS?

The answer is No, because unlike Hypervisors, Docker is not meant to virtualize and run different Operating Systems and Kernels on the same hardware. The main purpose of the Docker is to containerize applications and to ship them and run them.

**Virtual Machines vs Containers**

It is important to note that Docker has less isolation as more resources are shared in containers like the Kernel, whereas VMs have complete isolation from each other. Since VMs don’t rely on the underlying Operating System or Kernel, you can have different types of Operating Systems such as Linux, Windows on the same hypervisor, whereas it is not possible on a single Docker Host.



**Container vs Image**

An image is a package or a template, just like a VM template that you might have worked with, in the virtualization world. It is used to create one or more containers. Containers are running instances of images that are isolated and have their own environments and set of processes.

**Container advantages**

With Docker, setting up the infrastructure is in the hands of a developer in the form of a Docker file. Using the Docker file, a developer can create an image of the application, which can be used to run on any container platform and is guaranteed to run same way everywhere, which was not the case earlier without Docker. So, the OPS team can simply use the image to deploy the application.