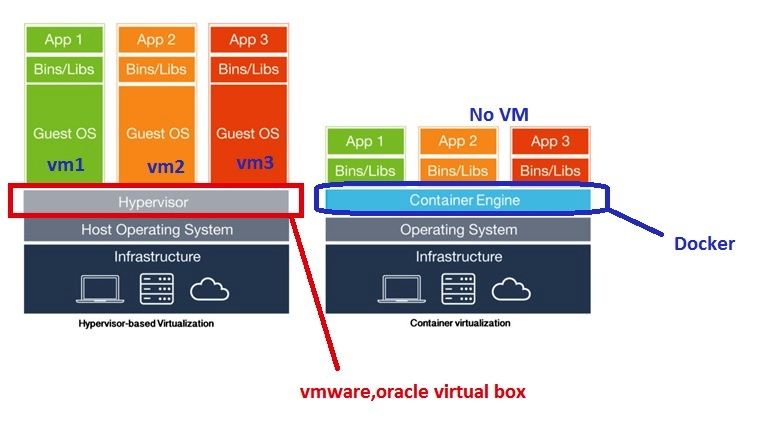
****

**Containers Introduction**

**What are “containers” and “VMs”?**

Containers and VMs are similar in their goals: to isolate an application and its dependencies into a self-contained unit that can run anywhere.

Moreover, containers and VMs remove the need for physical hardware, allowing for more efficient use of computing resources, both in terms of energy consumption and cost effectiveness

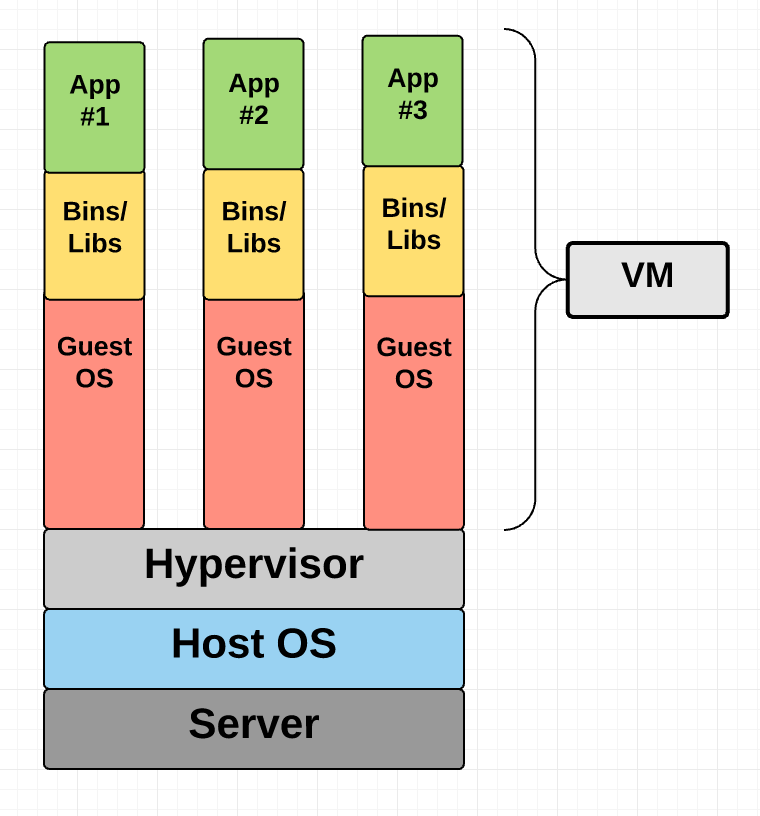


### Virtual Machines

A VM is essentially an emulation of a real computer that executes programs like a real computer. VMs run on top of a physical machine using a “hypervisor”. A hypervisor, in turn, runs on either a host machine or on “bare-metal”.

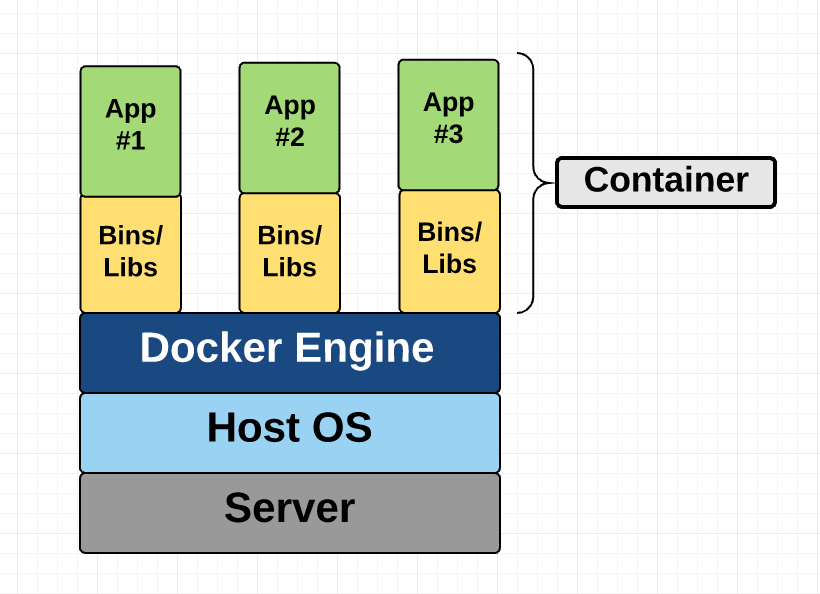
A **hypervisor** is a piece of software, firmware, or hardware that VMs run on top of. The hypervisors themselves run on physical computers, referred to as the “host machine”. The host machine provides the VMs with resources, including RAM and CPU. These resources are divided between VMs and can be distributed as you see fit. So if one VM is running a more resource heavy application, you might allocate more resources to that one than the other VMs running on the same host machine.

The VM that is running on the host machine (again, using a hypervisor) is also often called a “guest machine.” This guest machine contains both the application and whatever it needs to run that application (e.g. system binaries and libraries). It also carries an entire virtualized hardware stack of its own, including virtualized network adapters, storage, and CPU — which means it also has its own full-fledged guest operating system. From the inside, the guest machine behaves as its own unit with its own dedicated resources. From the outside, we know that it’s a VM — sharing resources provided by the host machine.



### Container

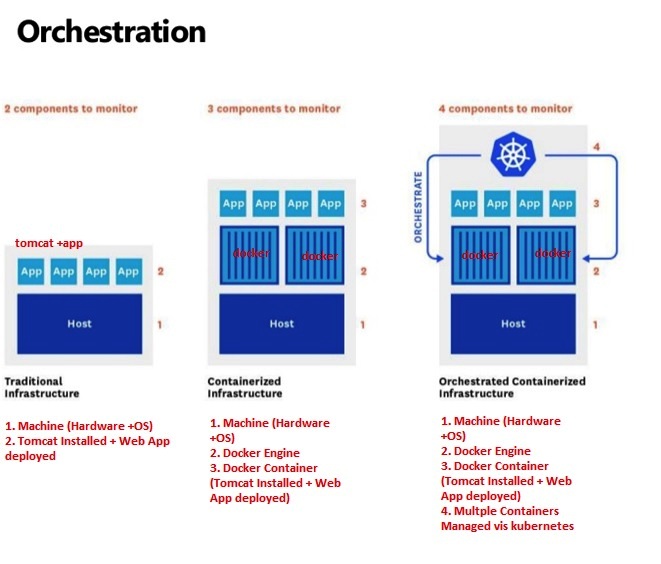
Unlike a VM which provides hardware virtualization, a container provides operating-system-level virtualization by abstracting the “user space”

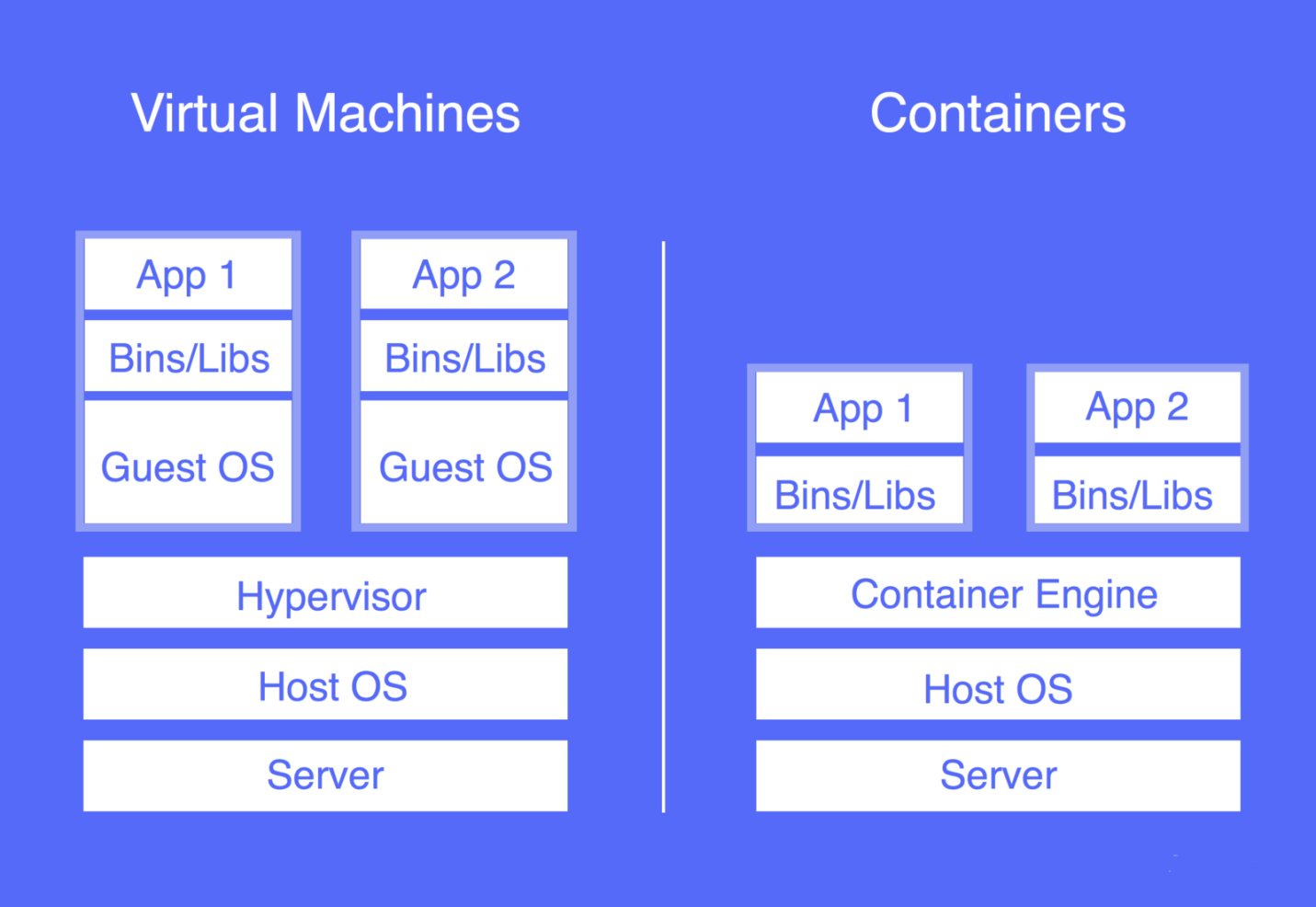


Items usually bundled into a container include:

* Application
* Dependencies
* Libraries
* Binaries
* Configuration files

Containerizing an application enables it to run reliably in different environments by abstracting away the operating system and the physical infrastructure. Containerized applications are sharing the kernel of the host operating system with other containers and the shared part of the OS is read only.





This diagram shows you that containers package up just the user space, and not the kernel or virtual hardware like a VM does. Each container gets its own isolated user space to allow multiple containers to run on a single host machine. We can see that all the operating system level architecture is being shared across containers. The only parts that are created from scratch are the bins and libs. This is what makes containers so lightweight

