Contenedores

Docker

- Docker is an elegant and beautiful way to package and run applications. Using your favorite Linux system, you can have Docker installed and running as a service in just a few minutes. The ease with which you can then build, run, stop, start, investigate, modify, and otherwise manipulate containers is, honestly, awesome.

- Docker’s ease of use contributed to it becoming one of the most popular open source projects today. But Docker as a centerpiece for containerization of the data center has caused the most commotion. The potential is not less than the reinvention of how individuals and companies, large and small, create, test, deploy, and manage their most critical applications.

- With containerization also comes the possibility of more efficiently deploying applications into cloud environments. Like containers themselves, the operating systems that run containers can be slimmed down. These new, container-ready host operating systems no longer have to carry all the dependencies that an application requires because the container is already holding most of what it needs to run

Pros and Cons of Containerizing Applications

- Understanding Pros and Cons of Containerizing Applications
- Docker provides a way to create and run applications that have been configured within a container. To truly understand what that means, it helps to start with what a containerized application is not. A containerized application is NOT......An Application Running Directly on a Host Computer
- The traditional way to run an application is to install and run that application directly from a host computer's file system. That application's view of its environment would include the host's process table, file system, IPC facility, network interfaces, ports, and devices.
- To get the application working, you often need to install additional software packages to go with your application. Normally, this is not a problem. But in some cases, you might want to run different versions of the same package running on the same system, which could cause conflicts.
- The application could conflict with applications in other ways as well. If the application is a service, it might bind to a particular network port by default. It might also read common configuration files when the service starts up. This could make it impossible, or at least tricky, to have multiple instances of that service running on the same host computer. It could also make it difficult to run other services that want to bind to that same port.

- Another downside of running an application directly on the host computer is that it can be difficult to move that application around. If the host computer needed to be shut down or if the application needed more capacity than is available on the host system, it might not be easy to pick up all the dependencies from the host computer and move them to another host.
- An Application Running Directly within a Virtual Machine
- Creating a virtual machine (VM) for the specific purpose of running an application can overcome some of the drawbacks of running applications directly on the host operating system. Although a virtual machine is on the host, it runs as a separate operating system, which includes its own kernel, file system, network interfaces, and so on. This makes it easy to keep almost everything inside the operating system separate from the host.
- Because a VM is a separate entity, you don’t have the same issues of inflexibility that come from running an application directly on hardware. You could run an application 10 times on the host by starting up 10 different VMs. The service on each VM could listen on the same port number, but not cause a conflict because each VM could have a different IP address.
- Likewise, if you need to shut down a host computer, you could either migrate the VM to another host (if your virtualization environment supports it) or just shut it down and start it again on the new host.

- The downside of running each instance of an application in a VM is the resources it consumes. Your application might require only a few megabytes of disk space to run, but the entire VM could consume many gigabytes of space. Also, the startup time and CPU consumption of the VM is almost sure to be higher than the application itself would consume.
- Containers offer an alternative to running applications directly on the host or in a VM that can make the applications faster, more portable, and more scalable.
- Understanding the Upside of Containers
- For running applications, containers offer the promise of both flexibility and efficient resource usage.
- Flexibility comes from the container being able to carry all the files it needs with it. Like the application running in a VM, it can have its own configuration files and dependent libraries, as well as having its own network interfaces that are distinct from those configured on the host. So, again, as with the VM, a containerized application should be able to move around more easily than its directly installed counterparts and not have to contend for the same port numbers because each container they run in has separate network interfaces.
- As for startup time and consumption of disk space and processing power, a container is neither running a separate operating system nor should it hold the amount of software needed to run a whole operating system. That’s because the container can contain just what the application needs to run, along with any other tools you might want to run with the container and a small amount of metadata describing the container.
- Docker containers don’t have a separate kernel, as a VM does. Commands run from a Docker container appear in the process table on the host and, in most ways, look much like any other process running on the system. The difference between an application run in those two environments, however, has most to do with the different view of the world those two applications have looking out:
- Image File system: The container has its own file system and cannot see the host system’s file system by default. One exception to this rule is that files (such as /etc/hosts and /etc/resolv.conf) may be automatically bind mounted inside the container. Another exception is that you can explicitly mount directories from the host inside the container when you run a container image.
- Image Process table: Hundreds of processes may be running on a Linux host computer. However, by default, processes inside a container cannot see the host’s process table, but instead have their own process table. So the application’s process you run when you start up the container is assigned PID 1 within the container. From inside the container, a process cannot see any other processes running on the host that were not launched inside the container.
- Image Network interfaces: By default, the Docker daemon defines an IP address via DHCP from a set of private IP addresses. Instead of using DHCP, Docker supports other network modes, such as allowing containers to use another container’s network interfaces, the host’s network interfaces directly, or no network interfaces. If you choose, you can expose a port from inside the container to the same or different port number on the host.
- Image IPC facility: Processes running inside containers cannot interact directly with the inter-process communications (IPC) facility running on the host system. You can expose the IPC facility on the host to the container, but that is not done by default. Each container has its own IPC facility.
- Image Devices: Processes inside the container cannot directly see devices on the host system. Again, a special privilege option can be set when the container is run to grant that privilege.
- As you can see, Docker containers have the capability to run in plain sight to the host, but in a way that restricts what the container can see outside its boundaries into the host (unless you explicitly open those views).

Challenges of Containerizing Applications

- Among the challenges of containerizing applications is the fact that they are different from applications not in a container. In every Linux system facilities are in place for starting and stopping services and viewing error messages. Linux also provides ways of monitoring services and rotating log files.
- For running virtual machines, whole virtualization platforms, such as OpenStack and Red Hat Enterprise Virtualization, are built to start, stop, and otherwise work with VMs. Although efforts are underway to build tools for managing sets of containers, most are still in their infancy. Frameworks for deploying and managing sets of containers are being put in place in projects such as Kubernetes and OpenShift.
- Docker containers are packaged as container images. Work has been done to be able to store container images in registries and manage them with the docker command. However, the tools for managing Docker images are not nearly as mature as those used to manage Linux software packages (such as those for Linux RPM or Deb based systems).
- Tools are just now being developed to be able to verify where an image came from, to determine whether it has been tampered with, and to see exactly which software packages and their versions have been installed in the container. For now, however, be aware that it is difficult in most cases to be completely assured that random images you grab from the Docker Hub Registry are safe to use.
- Another challenge to using containers comes from the fact that containers, by their nature, cannot see other containers by default. So, what about the times that you want your container to work closely with another container? For example, you might have a web server that you want to access your database server.
- Some of the solutions for getting containers to see each other are features in Docker that let you link containers together and Kubernetes features that let you identify services that are used and provided between containers in pods. More container management tools are also becoming available to deal with these issues. Just keep in mind that they are in early stages of development, and multiple, sometimes conflicting, tools are being developed in almost every area of container management.

What Makes Up Docker

- Docker is a container format developed by the Docker Project. The docker command can run, stop, start, investigate, and otherwise manipulate containers. The docker command also can run as a service daemon, handling requests to manage Docker containers. This Docker service, by default, grabs the images you request from the Docker Hub Registry. You don't need to know much more than that to get started, but some additional words are in order.
- The Docker Project
- The Docker Project (https://www.docker.com) provides a focal point for Docker development. It refers to Docker as "an open platform for developers and sysadmins of distributed applications." Its goal is to simplify application development and distribution.
- Solomon Hykes is the founder and CTO of Docker. He compares what Docker sets out to do in the software industry to what physical shipping containers have done for the shipping industry. Whether you are shipping cars, barrels, boxes, or pianos, by using a standard container to ship those diverse types of items, the tools you use for transporting and working with them can become standardized as well.
- So, at its core, the Docker Project provides a format for software containers and creates a simple infrastructure that is set up specifically to work with software in that format. As the project has progressed, it has begun extending out beyond its initial focus on stabilizing the Docker format and providing the tools to manage single containers.
- Today, the Docker Project is expanding its scope to include provisioning and orchestration tools, to help people deploy and manage groups of containers. It is also working on ways to manage computing resources and help run Docker containers in ways that offer high availability. As those tools become available, they will have to go head-to-head against more established container orchestration tools being developed by companies such as Google and Red Hat (tools that include the Kubernetes project covered in this book).
- For now, however, the Docker Project's greatest achievements are the Docker container format, the tools for managing individual containers, and the capability to pull and push Docker container images between Docker clients and registries. The central registry, which is managed by the Docker Project, is referred to as the Docker Hub Registry.
- The Docker Hub Registry
- The Docker Hub Registry (https://registry.hub.docker.com) offers a place where individuals and organizations can store and develop their Docker container images. When you install Docker on your Linux system, by default Docker looks to the Docker Hub Registry when you make requests for Docker container images not already on your system

Docker Images and Containers

- The goal of containerization is to gather together all the components an application needs to run in a single, contained unit. For Docker, that unit is referred to as a Docker image. Inside the image is the application the container is intended to execute and, typically, any libraries, configuration files, executables, or other components that the application needs to execute.

- An image is a static unit that sits in a repository, or the local file system where Docker is installed, and waits to run. When you save a Docker image to a file system, as opposed to storing it in a repository, it is stored as a tarball. That tarball can be transported as you would any other file and then imported later to run as a container on your local system running Docker.

- Major Linux distributions, such as Red Hat Enterprise Linux, Ubuntu, Fedora, and CentOS offer official base images that you can use to build your own Docker images. You don’t have to be a programmer to take a base image, add existing applications to it, and make it into your own images. You do this by creating a Dockerfile file and running a docker build command on it.

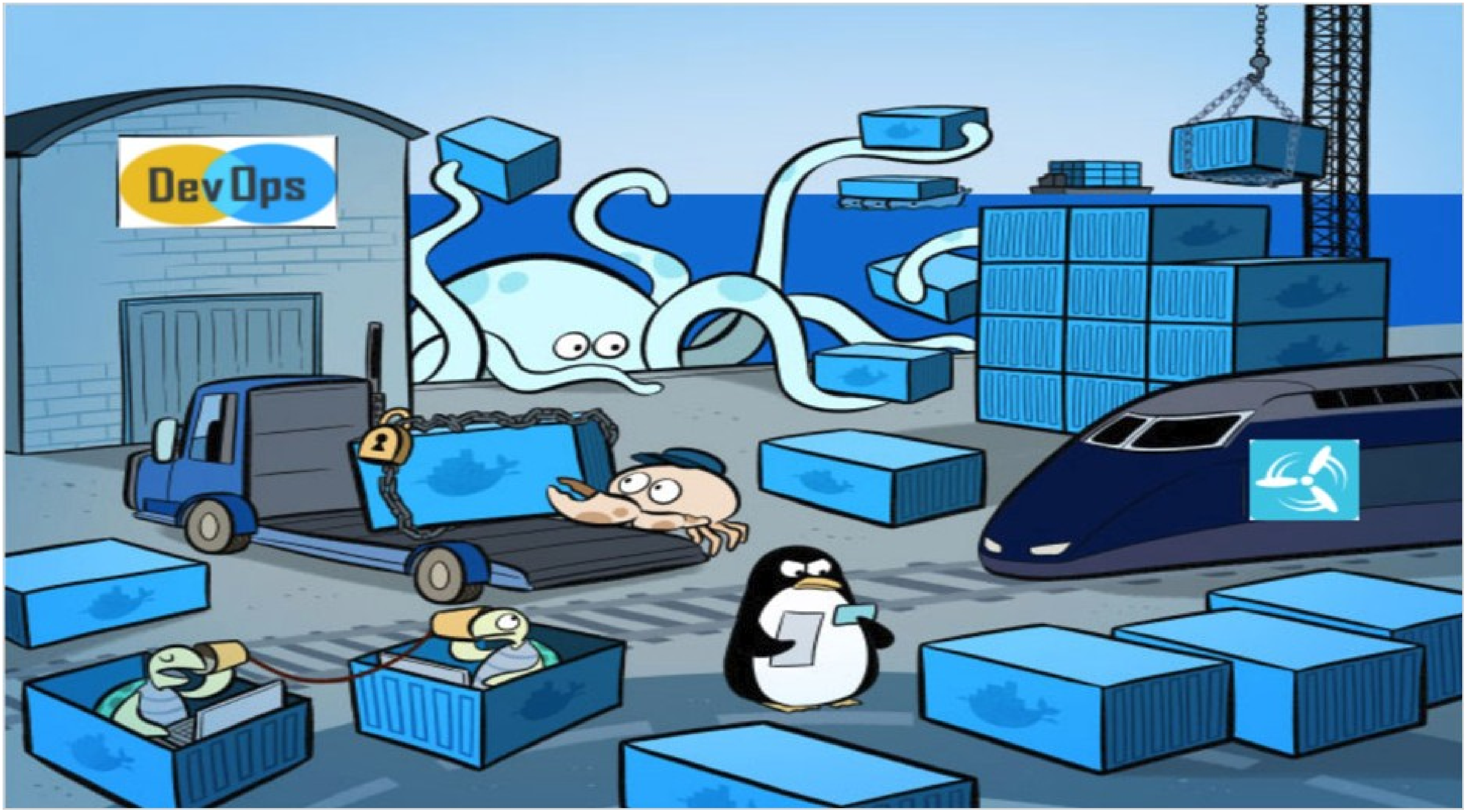
- The term Docker container refers to a running instance of a Docker image. Or, more precisely, an instance of an image that has run, since it may be running, paused, or stopped at the moment. The distinction between images and containers is critical when you start to use Docker. The reason you need to understand that distinction is that there are different commands for working with images versus working with containers.

Docker concepts

- Docker is a platform for developers and sysadmins to develop, deploy, and run applications with containers. The use of Linux containers to deploy applications is called containerization. Containers are not new, but their use for easily deploying applications is.
- Containerization is increasingly popular because containers are:
- Flexible: Even the most complex applications can be containerized.
- Lightweight: Containers leverage and share the host kernel.
- Interchangeable: You can deploy updates and upgrades on-the-fly.
- Portable: You can build locally, deploy to the cloud, and run anywhere.
- Scalable: You can increase and automatically distribute container replicas.
- Stackable: You can stack services vertically and on-the-fly.

Docker Versions

- Docker Enterprise Edition (Docker EE): This is designed for enterprise development and IT teams who build, ship, and run business-critical applications in production at scale. Docker EE is integrated, certified, and supported to provide enterprises with the most secure container platform in the industry to modernize all applications.
- Docker Community Edition (Docker CE): This is ideal for developers and small teams looking to get started with Docker and experimenting with container-based applications. Docker CE is available on many platforms, from desktop to cloud to the server. Docker CE is available for macOS and Windows and provides a native experience to help you focus on learning Docker. You can build and share containers and automate the development pipeline, all from a single environment.

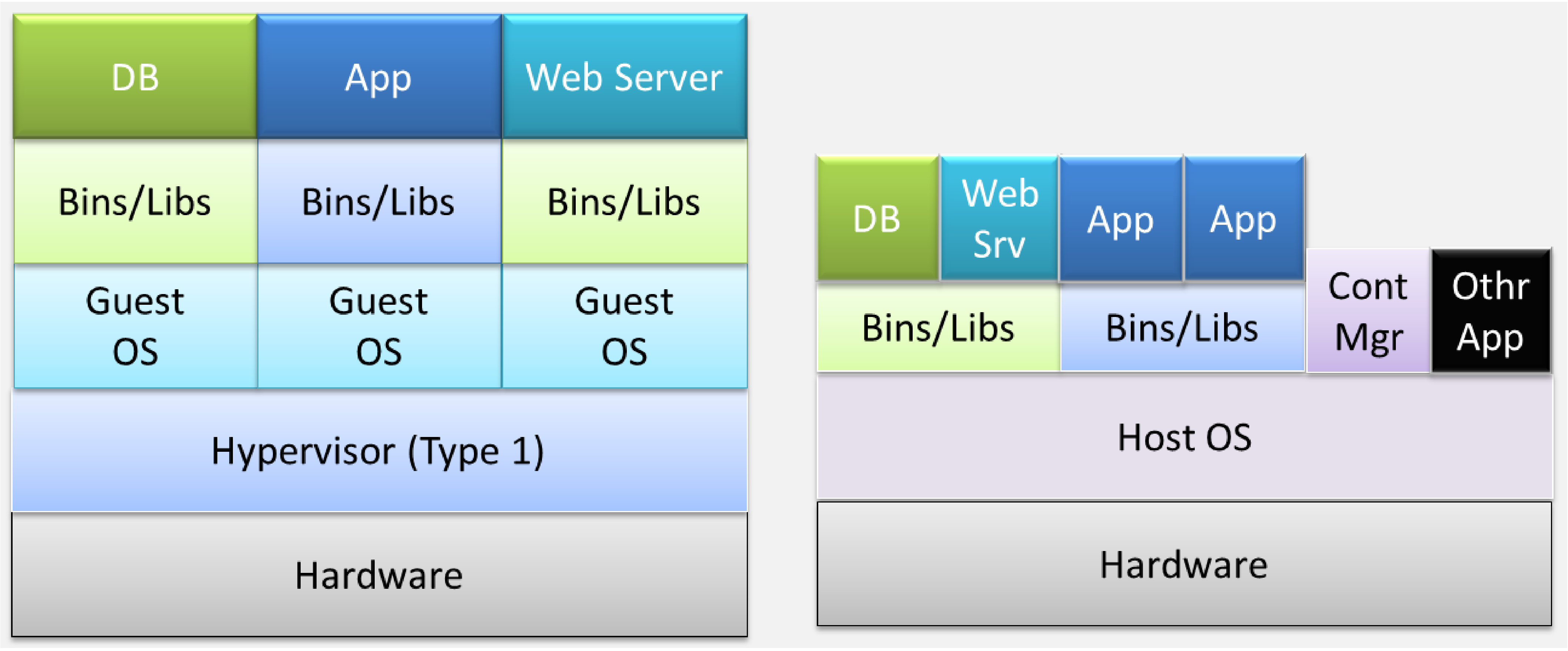

DevOps

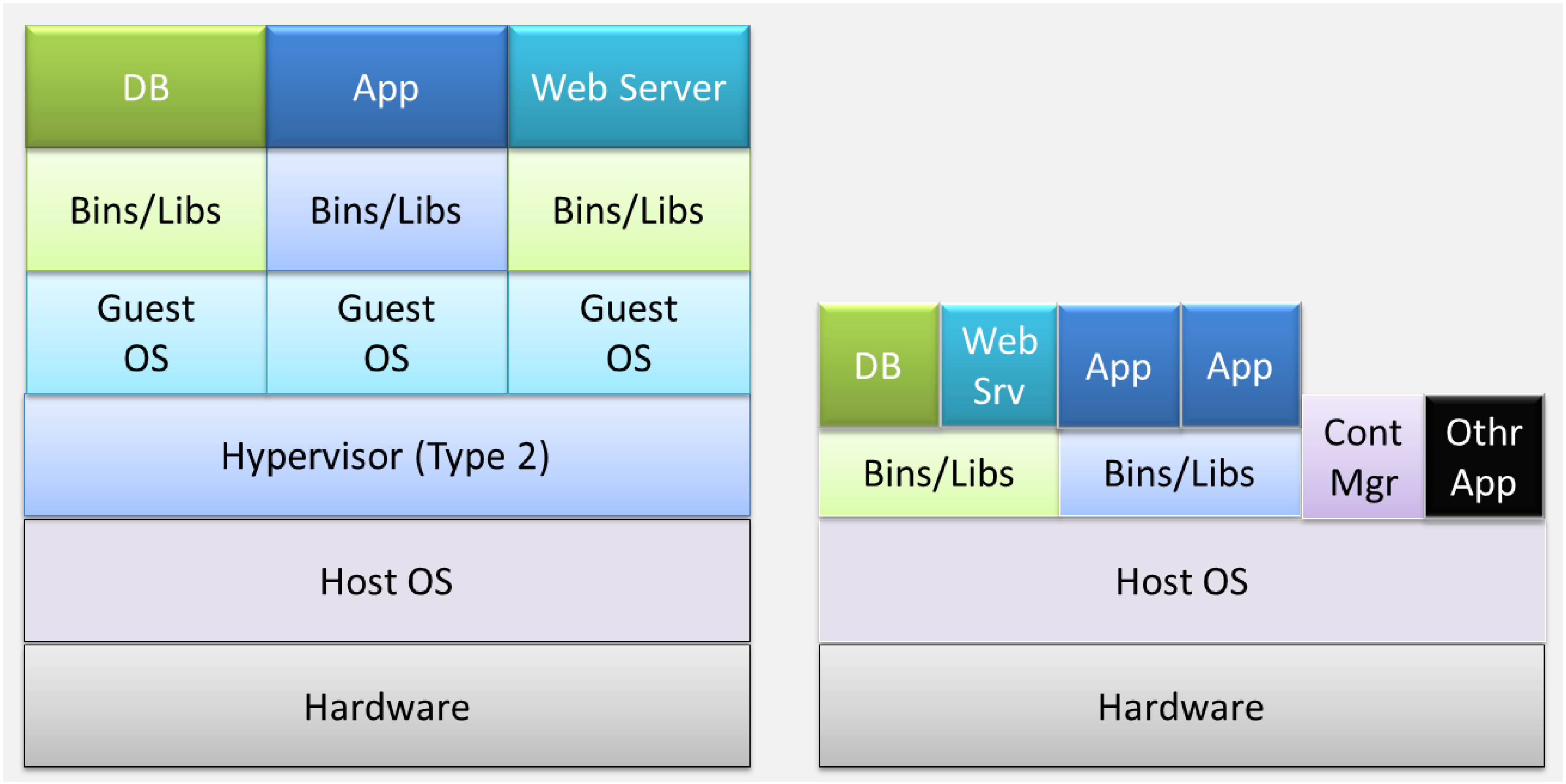
Images and containers

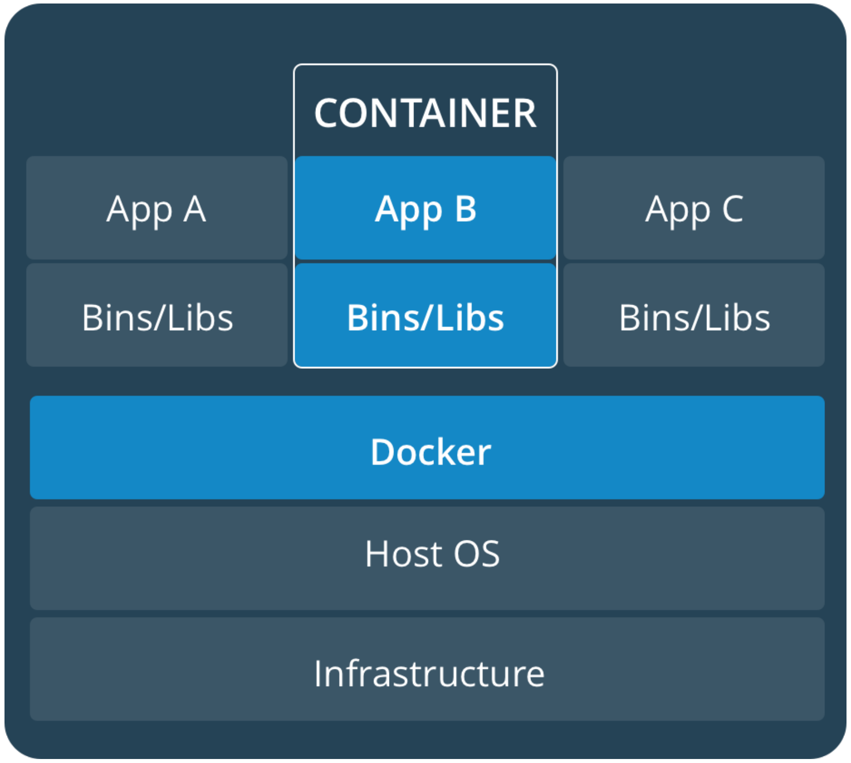
- A container is launched by running an image. An image is an executable package that includes everything needed to run an application--the code, a runtime, libraries, environment variables, and configuration files.
- A container is a runtime instance of an image--what the image becomes in memory when executed (that is, an image with state, or a user process). You can see a list of your running containers with the command, docker ps, just as you would in Linux.
- Containers and virtual machines
- A container runs natively on Linux and shares the kernel of the host machine with other containers. It runs a discrete process, taking no more memory than any other executable, making it lightweight.
- By contrast, a virtual machine (VM) runs a full-blown “guest” operating system with virtual access to host resources through a hypervisor. In general, VMs provide an environment with more resources than most applications need.

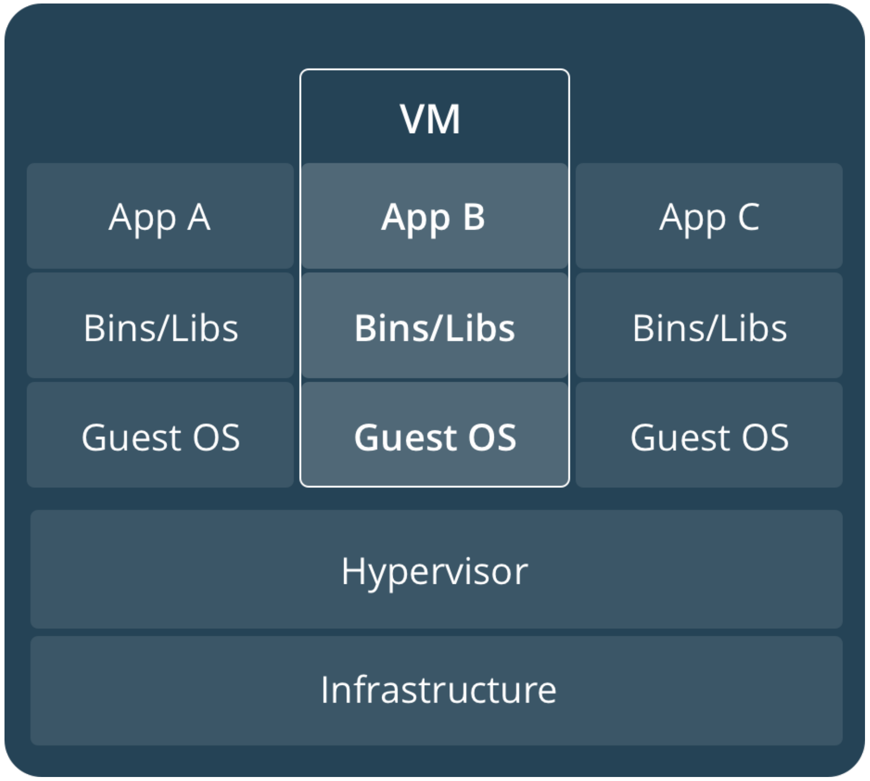
Images and containers

- A container is launched by running an image. An image is an executable package that includes everything needed to run an application--the code, a runtime, libraries, environment variables, and configuration files.
- A container is a runtime instance of an image--what the image becomes in memory when executed (that is, an image with state, or a user process). You can see a list of your running containers with the command, docker ps, just as you would in Linux.
- Containers and virtual machines
- A container runs natively on Linux and shares the kernel of the host machine with other containers. It runs a discrete process, taking no more memory than any other executable, making it lightweight.
- By contrast, a virtual machine (VM) runs a full-blown “guest” operating system with virtual access to host resources through a hypervisor. In general, VMs provide an environment with more resources than most applications need.


DB
App
Web Server
Bins/Libs
Bins/Libs
Bins/Libs
Guest OS
Guest OS
Guest OS
Hypervisor (Type 1)
Hardware
DB
Web Srv
App
App
Bins/Libs
Bins/Libs
Cont Mgr
Othr App
Host OS
Hardware


DB
App
Web Server
Bins/Libs
Bins/Libs
Bins/Libs
Guest OS
Guest OS
Guest OS
Hypervisor (Type 2)
Host OS
Hardware
DB
Web Srv
App
App
Bins/Libs
Bins/Libs
Cont Mgr
Othr App
Host OS
Hardware


CONTAINER
App A
App B
App C
Bins/Libs
Bins/Libs
Bins/Libs
Docker
Host OS
Infrastructure


VM
App A
App B
App C
Bins/Libs
Bins/Libs
Bins/Libs
Guest OS
Guest OS
Guest OS
Hypervisor
Infrastructure

Instalar y probar Docker

Instalación

- sudo apt-get remove docker docker-engine docker.io
- sudo apt-get install \

```
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg2 \
    software-properties-common
```

- curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key add -
- sudo apt-key fingerprint 0EBFCD88
- sudo add-apt-repository \

  "deb [arch=amd64] https://download.docker.com/linux/debian \

  $(lsb_release -cs) \

  stable"

- sudo apt-get install docker-ce
- apt-cache madison docker-ce
- sudo apt-get install dockerce=<VERSION_STRING>
- docker info
- docker pull hello-world
- docker images

- docker run hello-world
- mkdir miapache
- mkdir public-html
- vi public-html/index.html
- vi Dockerfile

  ```
  FROM httpd:2.4
  COPY ./public-html/ /usr/local/apache2/htdocs/
  ```
- docker build -t my-apache .
- docker run -dit --name my-running-app -p 8080:80 my-apache
- Acceder a localhost:8080