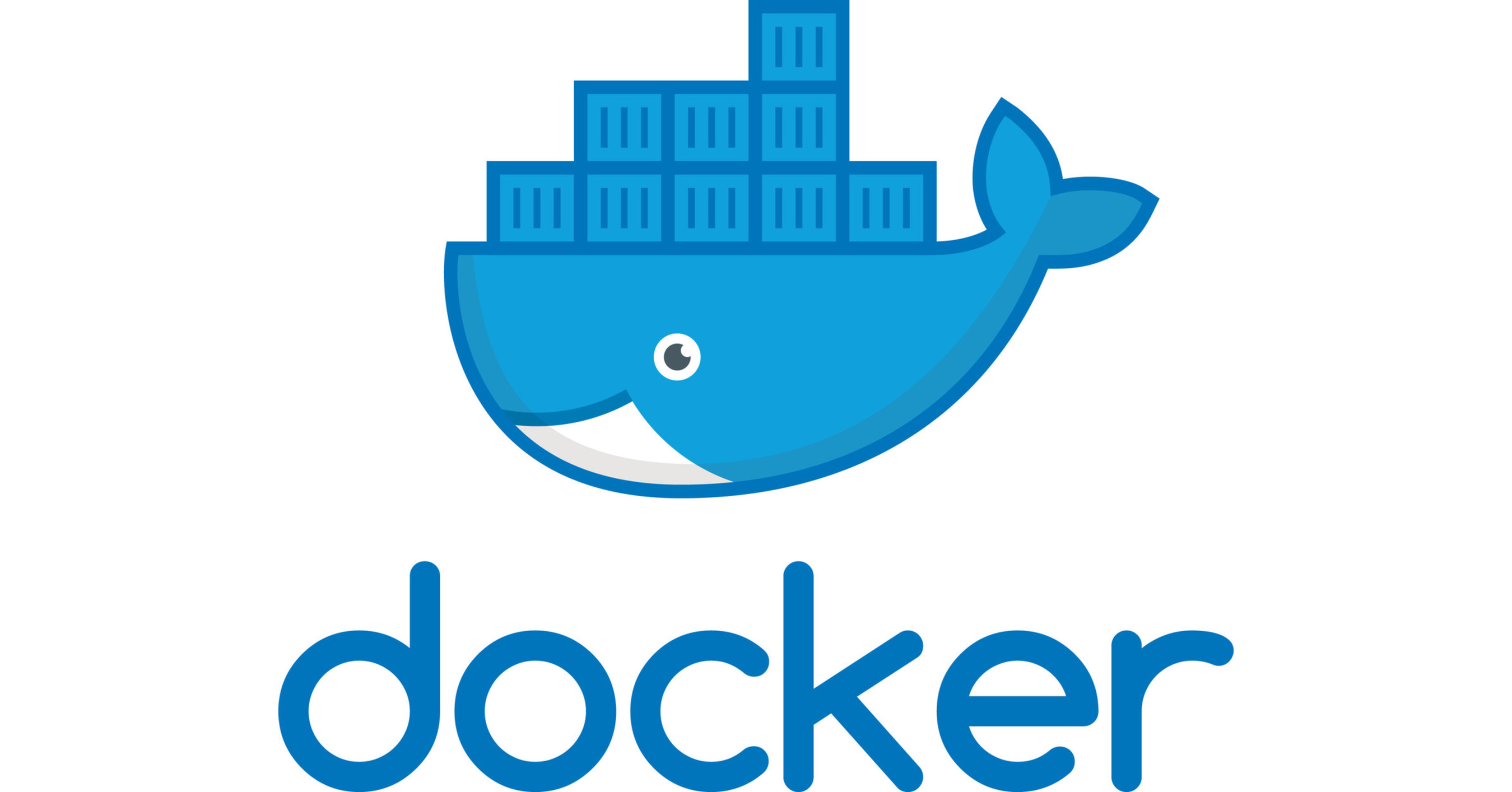
**Docker**





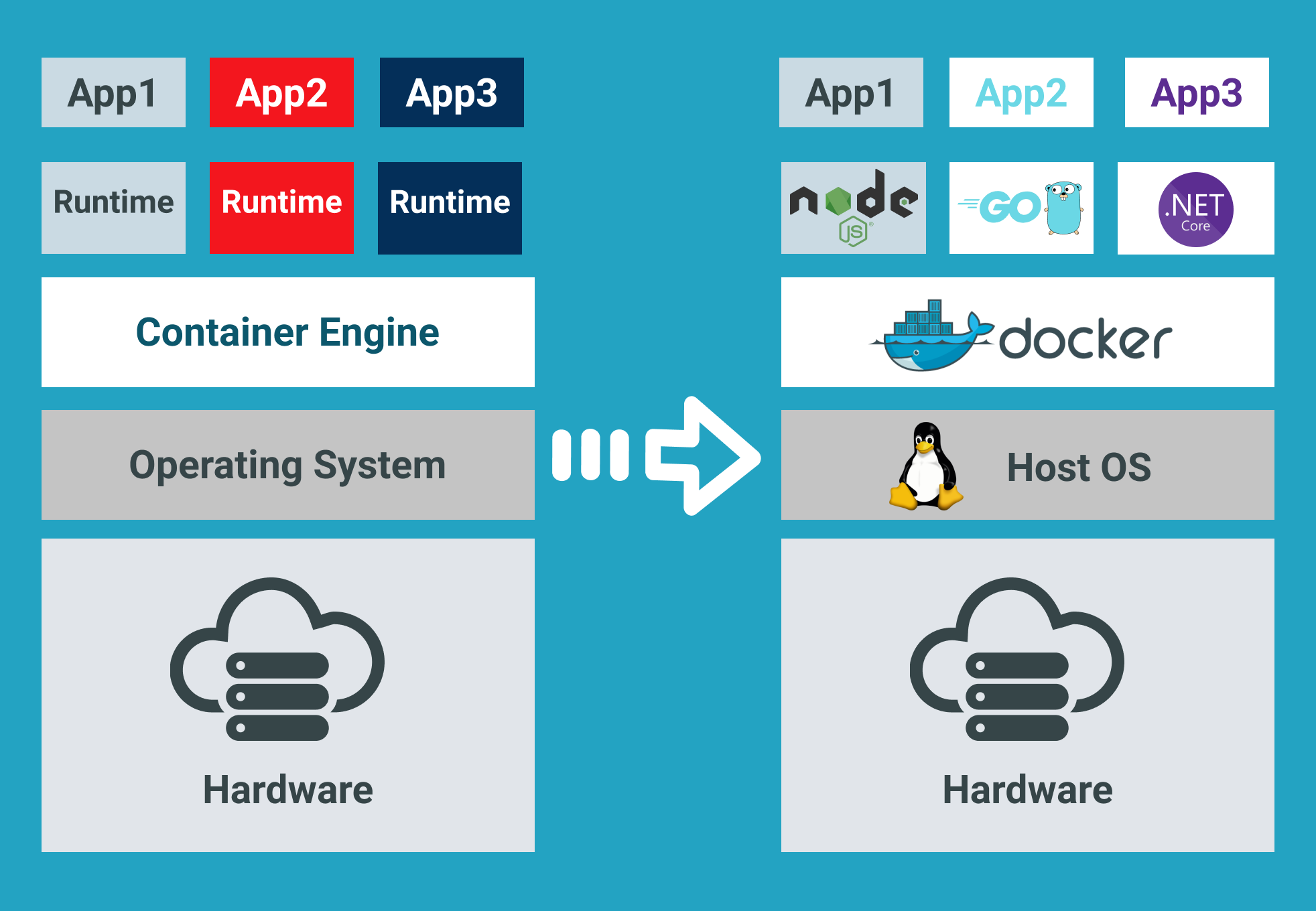
## **What Is Docker?**

Docker is an open-source containerization platform used for developing, deploying, and managing applications in lightweight virtualized environments called containers.

It is mainly used as a software development platform for developing [distributed applications](https://phoenixnap.com/blog/securely-connect-distributed-apps) that work efficiently in different environments. By making the software system agnostic, developers don’t have to worry about compatibility issues. Packaging apps into isolated environments (containers) also makes it easier to develop, deploy, maintain, and use applications.

## Features of Docker

* Docker has the ability to reduce the size of development by providing a smaller footprint of the operating system via containers.
* With containers, it becomes easier for teams across different units, such as development, QA and Operations to work seamlessly across applications.
* You can deploy Docker containers anywhere, on any physical and virtual machines and even on the cloud.
* Since Docker containers are pretty lightweight, they are very easily scalable.



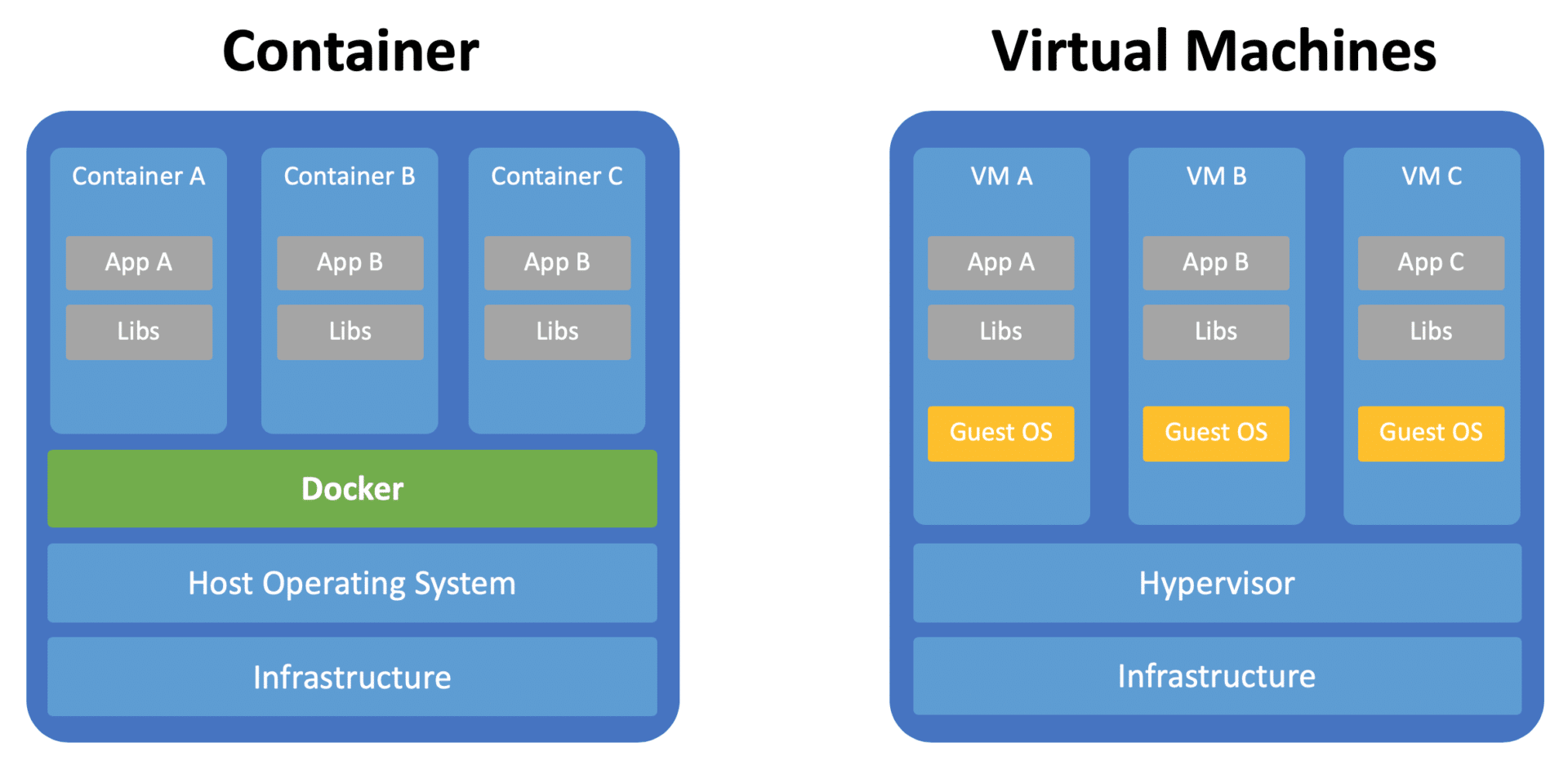
## Components of Docker

Docker has the following components

* **Docker for Mac** − It allows one to run Docker containers on the Mac OS.
* **Docker for Linux** − It allows one to run Docker containers on the Linux OS.
* **Docker for Windows** − It allows one to run Docker containers on the Windows OS.
* **Docker Engine** − It is used for building Docker images and creating Docker containers.
* **Docker Hub** − This is the registry which is used to host various Docker images.
* **Docker Compose** − This is used to define applications using multiple Docker containers.

## **What Are Containers?**

Docker containers are lightweight virtualized runtime environments for running applications. Each container represents a package of software that contains code, system tools, runtime, libraries, dependencies, and configuration files required for running a specific application. They are independent and isolated from the host and other instances running on the host.



# **Installing Docker**

<https://docs.docker.com/engine/install/>

# **Docker - Hub**

Docker Hub is a registry service on the cloud that allows you to download Docker images that are built by other communities. You can also upload your own Docker built images to Docker hub. In this chapter, we will see how to download and the use the Jenkins Docker image from Docker hub.

<https://hub.docker.com/>

# **Docker - Images**

In Docker, everything is based on Images. An image is a combination of a file system and parameters

docker run hello-world

* The Docker command is specific and tells the Docker program on the Operating System that something needs to be done.
* The **run** command is used to mention that we want to create an instance of an image, which is then called a **container**.
* Finally, "hello-world" represents the image from which the container is made.

# **Docker - Containers**

Containers are instances of Docker images that can be run using the Docker run command. The basic purpose of Docker is to run containers. Let’s discuss how to work with containers.

## Running a Container

Running of containers is managed with the Docker **run** command. To run a container in an interactive mode, first launch the Docker container.

sudo docker run –it centos /bin/bash

# **Docker - Working with Containers**

## ocker top

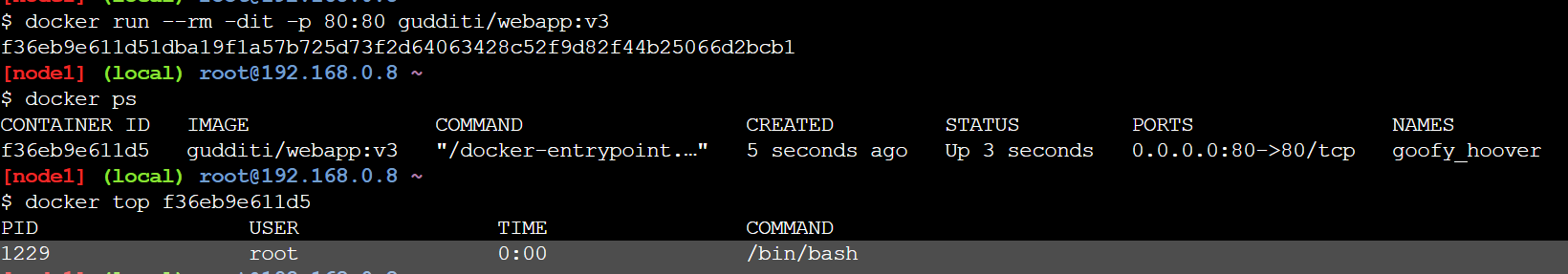
With this command, you can see the top processes within a container.

### Syntax

docker top ContainerID

### Options

* **ContainerID** − This is the Container ID for which you want to see the top processes.



|  |
| --- |
| Quick list of Docker Commands  * docker version – Echoes Client’s and Server’s Version of Docker * docker images – List all Docker images * docker build <image> – Builds an image form a Docker file * docker save <path> <image> – Saves Docker image to .tar file specified by path * docker run – Runs a command in a new container. * docker start – Starts one or more stopped containers * docker stop <container\_id> – Stops container * docker rmi <image> – Removes Docker image * docker rm <container\_id> – Removes Container * docker pull – Pulls an image or a repository from a registry * docker push – Pushes an image or a repository to a registry * docker export – Exports a container’s filesystem as a tar archive * docker exec – Runs a command in a run-time container * docker ps – Show running containers * docker ps -a – Show all containers * docker ps -l – Show latest created container * docker search – Searches the Docker Hub for images * docker attach – Attaches to a running container * docker commit – Creates a new image from a container’s changes |

# **Docker - File**

Docker also gives you the capability to create your own Docker images, and it can be done with the help of **Docker Files**. A Docker File is a simple text file with instructions on how to build your images.

**Step 1** − Create a file called **Docker File** and edit it using **vim**. Please note that the name of the file has to be "Dockerfile" with "D" as capital.

**Step 2** − Build your Docker File using the following instructions.

# **#This is a sample Image** **FROM** gudditi/webapp:v3 **MAINTAINER gudditinaganjaneyulu@gmail.com** **RUN apt-get update** **RUN apt-get install –y nginx** **CMD [“echo”,”Image created”]**

The following points need to be noted about the above file −

* The first line "#This is a sample Image" is a comment. You can add comments to the Docker File with the help of the **#** command
* The next line has to start with the **FROM** keyword. It tells docker, from which base image you want to base your image from. In our example, we are creating an image from the **ubuntu** image.
* The next command is the person who is going to maintain this image. Here you specify the **MAINTAINER** keyword and just mention the email ID.
* The **RUN** command is used to run instructions against the image. In our case, we first update our Ubuntu system and then install the nginx server on our **ubuntu** image.
* The last command is used to display a message to the user.

**Step 3** − Save the file.

# **Docker - Building Files**

Docker File in the last chapter. It’s now time to build the Docker File. The Docker File can be built with the following command −

docker build

Let’s learn more about this command.

## docker build

This method allows the users to build their own Docker images.

### Syntax

docker build -t ImageName:TagName dir

### Options

* **-t** − is to mention a tag to the image
* **ImageName** − This is the name you want to give to your image.
* **TagName** − This is the tag you want to give to your image.
* **Dir** − The directory where the Docker File is present.

### Return Value

None

### Example

sudo docker build –t myimage:0.1.

# **Docker - Managing Ports**

In Docker, the containers themselves can have applications running on ports. When you run a container, if you want to access the application in the container via a port number, you need to map the port number of the container to the port number of the Docker host. Let’s look at an example of how this can be achieved.

# docker run --rm -dit –name webapp -p 80:80 gudditi/webapp:v3

docker inspect webapp

# **Docker - Container Linking**

Container Linking allows multiple containers to link with each other. It is a better option than exposing ports. Let’s go step by step and learn how it works.

We can achieve using networking

# **Docker - Storage**

Docker has multiple storage drivers that allow one to work with the underlying storage devices. The following table shows the different storage drivers along with the technology used for the storage drivers.

|  |  |
| --- | --- |
| **Technology** | **Storage Driver** |
| OverlayFS | overlay or overlay2 |
| AUFS | aufs |
| Btrfs | brtfs |
| Device Manager | devicemanager |
| VFS | vfs |
| ZFS | zfs |

## Creating a Volume

A volume can be created beforehand using the **docker** command. Let’s learn more about this command.

### Syntax

docker volume create –-name=volumename –-opt options

### Options

* **name** − This is the name of the volume which needs to be created.
* **opt** − These are options you can provide while creating the volume.

### Return Value

The command will output the name of the volume created.

## Listing all the Volumes

You can also list all the **docker volumes** on a **docker host**. More details on this command is given below −

### Syntax

docker volume ls

# **Docker - Networking**

Docker takes care of the networking aspects so that the containers can communicate with other containers and also with the Docker Host. If you do an **ifconfig** on the Docker Host, you will see the Docker Ethernet adapter. This adapter is created when Docker is installed on the Docker Host.

Creating network

Docker network create sample

## Listing All Docker Networks

This command can be used to list all the networks associated with Docker on the host.

### Syntax

docker network ls

## Inspecting a Docker network

If you want to see more details on the network associated with Docker, you can use the Docker **network inspect** command.

### Syntax

docker network inspect networkname

# **Docker - Logging**

Docker has logging mechanisms in place which can be used to debug issues as and when they occur. There is logging at the **daemon level** and at the **container level**. Let’s look at the different levels of logging.

## Daemon Logging

At the daemon logging level, there are four levels of logging available −

* **Debug** − It details all the possible information handled by the daemon process.
* **Info** − It details all the errors + Information handled by the daemon process.
* **Errors** − It details all the errors handled by the daemon process.
* **Fatal** − It only details all the fatal errors handled by the daemon process.

Go through the following steps to learn how to enable logging.

**Step 1** − First, we need to stop the **docker daemon process**, if it is already running. It can be done using the following command −

sudo service docker stop

Docker Daemon Process

**Step 2** − Now we need to start the **docker daemon process**. But this time, we need to append the **–l** parameter to specify the logging option. So let’s issue the following command when starting the **docker daemon process**.

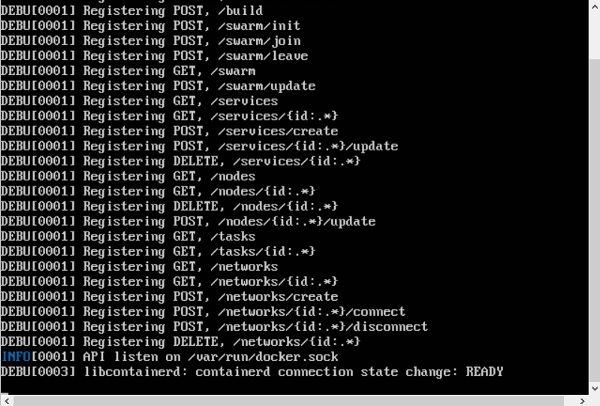
sudo dockerd –l debug &

The following points need to be noted about the above command −

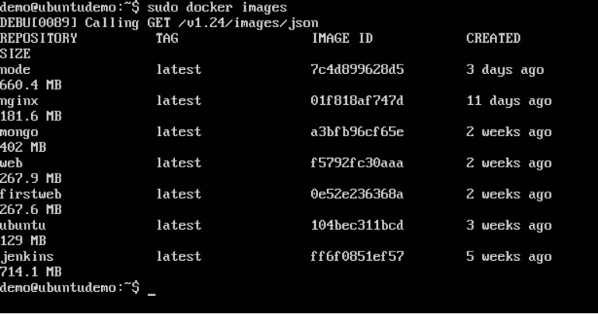
* **dockerd** is the executable for the **docker daemon process**.
* The **–l** option is used to specify the logging level. In our case, we are putting this as debug
* **&** is used to come back to the command prompt after the logging has been enabled.

Points

Once you start the Docker process with logging, you will also now see the **Debug Logs** being sent to the console.



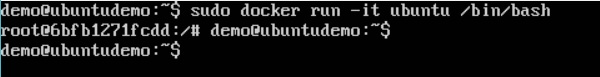
Now, if you execute any Docker command such as **docker images**, the Debug information will also be sent to the console.



## Container Logging

Logging is also available at the container level. So in our example, let’s spin up an Ubuntu container first. We can do it by using the following command.

sudo docker run –it ubuntu /bin/bash



Now, we can use the **docker log command** to see the logs of the container.

### Syntax

Docker logs containerID

### Parameters

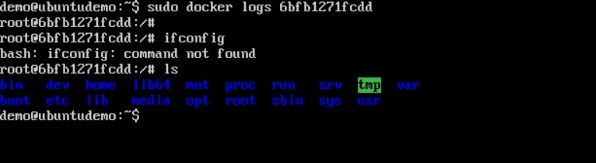
* **containerID** − This is the ID of the container for which you need to see the logs.

### Example

On our Docker Host, let’s issue the following command. Before that, you can issue some commands whilst in the container.

sudo docker logs 6bfb1271fcdd

### Output



# **Docker - Compose**

**Docker Compose** is used to run multiple containers as a single service. For example, suppose you had an application which required NGNIX and MySQL, you could create one file which would start both the containers as a service without the need to start each one separately.

In this chapter, we will see how to get started with Docker Compose. Then, we will look at how to get a simple service with MySQL and NGNIX up and running using Docker Compose.

Docker all concepts sample application code   
<https://github.com/GudditiNaganjaneyulu/DockerLoginApp>

U can clone that in a docker environment and run it using the below cmd

**docker-compose up --build –d**

Can browse that application using

http://<ip address>:8888

http://<ip address>:9999

------------------------------------------------------------------------------------------------------------------------------------------**Interview point questions:**