Docker overview

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code, you can significantly reduce the delay between writing code and running it in production.

[The Docker platform](https://docs.docker.com/get-started/overview/#the-docker-platform)

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security lets you run many containers simultaneously on a given host. Containers are lightweight and contain everything needed to run the application, so you don't need to rely on what's installed on the host. You can share containers while you work, and be sure that everyone you share with gets the same container that works in the same way.

Docker provides tooling and a platform to manage the lifecycle of your containers:

* Develop your application and its supporting components using containers.
* The container becomes the unit for distributing and testing your application.
* When you're ready, deploy your application into your production environment, as a container or an orchestrated service. This works the same whether your production environment is a local data center, a cloud provider, or a hybrid of the two.

[What can I use Docker for?](https://docs.docker.com/get-started/overview/#what-can-i-use-docker-for)

[Fast, consistent delivery of your applications](https://docs.docker.com/get-started/overview/#fast-consistent-delivery-of-your-applications)

Docker streamlines the development lifecycle by allowing developers to work in standardized environments using local containers which provide your applications and services. Containers are great for continuous integration and continuous delivery (CI/CD) workflows.

Consider the following example scenario:

* Your developers write code locally and share their work with their colleagues using Docker containers.
* They use Docker to push their applications into a test environment and run automated and manual tests.
* When developers find bugs, they can fix them in the development environment and redeploy them to the test environment for testing and validation.
* When testing is complete, getting the fix to the customer is as simple as pushing the updated image to the production environment.

[Responsive deployment and scaling](https://docs.docker.com/get-started/overview/#responsive-deployment-and-scaling)

Docker's container-based platform allows for highly portable workloads. Docker containers can run on a developer's local laptop, on physical or virtual machines in a data center, on cloud providers, or in a mixture of environments.

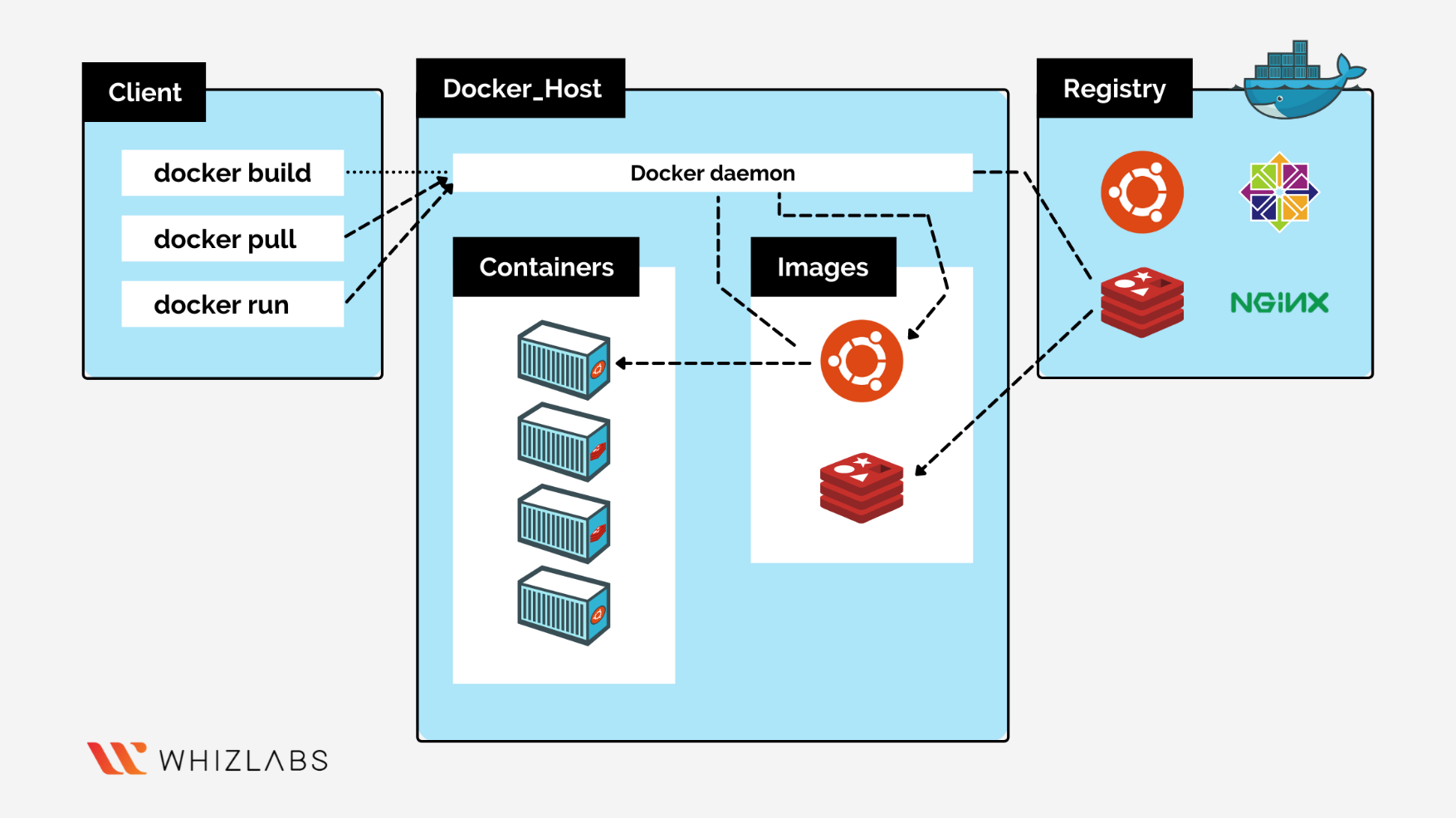
Docker's portability and lightweight nature also make it easy to dynamically manage workloads, scaling up or tearing down applications and services as business needs dictate, in near real time.

[Running more workloads on the same hardware](https://docs.docker.com/get-started/overview/#running-more-workloads-on-the-same-hardware)

Docker is lightweight and fast. It provides a viable, cost-effective alternative to hypervisor-based virtual machines, so you can use more of your server capacity to achieve your business goals. Docker is perfect for high density environments and for small and medium deployments where you need to do more with fewer resources.

[Docker architecture](https://docs.docker.com/get-started/overview/#docker-architecture)

Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface. Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers.



[The Docker daemon](https://docs.docker.com/get-started/overview/#the-docker-daemon)

The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

[The Docker client](https://docs.docker.com/get-started/overview/#the-docker-client)

The Docker client (docker) is the primary way that many Docker users interact with Docker. When you use commands such as docker run, the client sends these commands to dockerd, which carries them out. The docker command uses the Docker API. The Docker client can communicate with more than one daemon.

[Docker Desktop](https://docs.docker.com/get-started/overview/#docker-desktop)

Docker Desktop is an easy-to-install application for your Mac, Windows or Linux environment that enables you to build and share containerized applications and microservices. Docker Desktop includes the Docker daemon (dockerd), the Docker client (docker), Docker Compose, Docker Content Trust, Kubernetes, and Credential Helper. For more information, see [Docker Desktop](https://docs.docker.com/desktop/).

[Docker registries](https://docs.docker.com/get-started/overview/#docker-registries)

A Docker registry stores Docker images. Docker Hub is a public registry that anyone can use, and Docker looks for images on Docker Hub by default. You can even run your own private registry.

When you use the docker pull or docker run commands, Docker pulls the required images from your configured registry. When you use the docker push command, Docker pushes your image to your configured registry.

[Docker objects](https://docs.docker.com/get-started/overview/#docker-objects)

When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects. This section is a brief overview of some of those objects.

[Images](https://docs.docker.com/get-started/overview/#images)

An image is a read-only template with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization. For example, you may build an image which is based on the ubuntu image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.

You might create your own images, or you might only use those created by others and published in a registry. To build your own image, you create a Docker file with a simple syntax for defining the steps needed to create the image and run it. Each instruction in a Docker file creates a layer in the image. When you change the Docker file and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.

[Containers](https://docs.docker.com/get-started/overview/#containers)

A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.

By default, a container is relatively well isolated from other containers and its host machine. You can control how isolated a container's network, storage, or other underlying subsystems are from other containers or from the host machine.

A container is defined by its image as well as any configuration options you provide to it when you create or start it. When a container is removed, any changes to its state that aren't stored in persistent storage disappear.

[Example docker run command](https://docs.docker.com/get-started/overview/#example-docker-run-command)

The following command runs an ubuntu container, attaches interactively to your local command-line session, and runs /bin/bash.

$ docker run -i -t ubuntu /bin/bash

When you run this command, the following happens (assuming you are using the default registry configuration):

1. If you don't have the ubuntu image locally, Docker pulls it from your configured registry, as though you had run docker pull ubuntu manually.
2. Docker creates a new container, as though you had run a docker container create command manually.
3. Docker allocates a read-write filesystem to the container, as its final layer. This allows a running container to create or modify files and directories in its local filesystem.
4. Docker creates a network interface to connect the container to the default network, since you didn't specify any networking options. This includes assigning an IP address to the container. By default, containers can connect to external networks using the host machine's network connection.
5. Docker starts the container and executes /bin/bash. Because the container is running interactively and attached to your terminal (due to the -i and -t flags), you can provide input using your keyboard while Docker logs the output to your terminal.
6. When you run exit to terminate the /bin/bash command, the container stops but isn't removed. You can start it again or remove it.

[The underlying technology](https://docs.docker.com/get-started/overview/#the-underlying-technology)

Docker is written in the [Go programming languageopen\_in\_new](https://golang.org/) and takes advantage of several features of the Linux kernel to deliver its functionality. Docker uses a technology called namespaces to provide the isolated workspace called the container. When you run a container, Docker creates a set of namespaces for that container.

These namespaces provide a layer of isolation. Each aspect of a container runs in a separate namespace and its access is limited to that namespace.

**The Docker Platform**

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security let you to run many containers simultaneously on a given host. Containers are lightweight and contain everything needed to run the application, so you don't need to rely on what's installed on the host. You can share containers while you work and be sure that everyone you share with gets the same container that works in the same way.

***Uses of Docker***

**Fast consistent delivery of application**  
Docker provides tooling and a platform to manage the lifecycle of your containers:

Develop your application and its supporting components using containers.  
The container becomes the unit for distributing and testing your application.  
When you're ready, deploy your application into your production environment, as a container or an orchestrated service. This works the same whether your production environment is a local data center, a cloud provider, or a hybrid of the two.

Docker streamlines the development lifecycle by allowing developers to work in standardized environments using local containers which provide your applications and services. Containers are great for continuous integration and continuous delivery (CI/CD) workflows.

Consider the following example scenario:

* Your developers write code locally and share their work with their colleagues using Docker containers.
* They use Docker to push their applications into a test environment and run automated and manual tests.
* When developers find bugs, they can fix them in the development environment and redeploy them to the test environment for testing and validation.
* When testing is complete, getting the fix to the customer is as simple as pushing the updated image to the production environment.

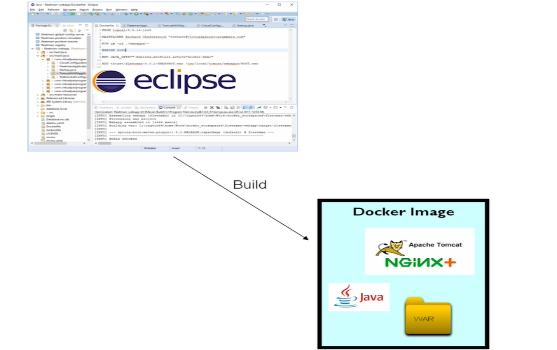
**Responsive Deployment and Scaling**  
Docker's container-based platform allows for highly portable workloads. Docker containers can run on a developer's local laptop, on physical or virtual machines in a data centre, on cloud providers, or in a mixture of environments.

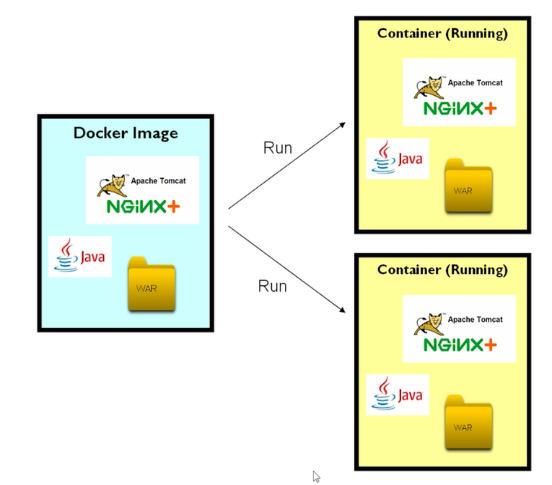
Docker's portability and lightweight nature also make it easy to dynamically manage workloads, scaling up or tearing down applications and services as business needs dictate, in near real time.

**Running more workload on same hardware**  
Docker is lightweight and fast. It provides a viable, cost-effective alternative to hypervisor-based virtual machines, so you can use more of your server capacity to achieve your business goals. Docker is perfect for high density environments and for small and medium deployments where you need to do more with fewer resources.

**Udemy Course:**

Get real world, hands-on experience with Docker - deploy a Java Microservice Architecture using Docker and Docker Swarm

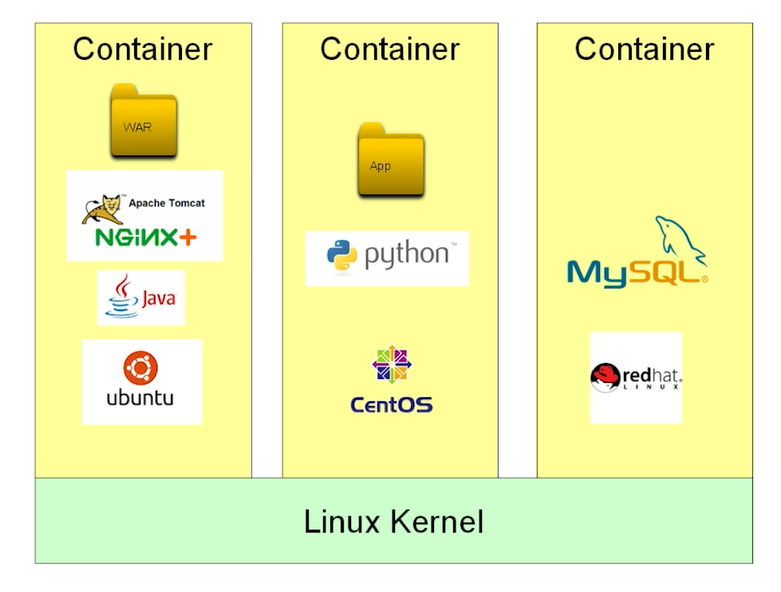
  
Before we are using a container first, we need build a docker image, image is a definition of a container, so where previously a developer is building a jar file or a war file a development environment such as eclipse spring boot and IntelliJ.   
  
When we are working on the docker style instead a developer is going to build an image and that image as describe previously is a definition of a complete environment.   
we can use standard java tools to build the images we can very easily MAVEN even for example create a docker image.



Somebody is deploying this application is live running server and we can run the docker image and is when we run image this will become a container, so container is instance of an image and instantiate one of the images by complete running it.

So, image is the entity we build, and we run that image it will become a container or runtime.

So, we could take a single image and you could run multiple times using the different containers.

We can develop a docker on Windows, Linux, and Mac, when we are launching a container, the container is running on top of our host operating system, the container is actually just the process running on our operating system.

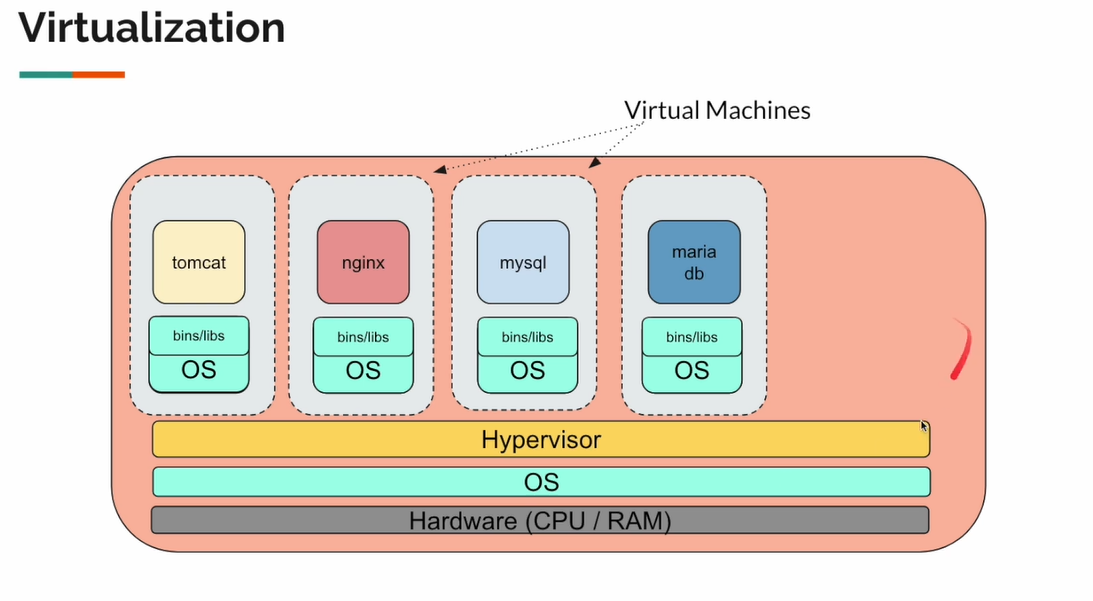
A container is not a virtual machine because it doesn’t contain its own operating system. Each container does contain its own distribution.

A container is much lighter, and we can run many more of them, so as a technical level the container is not a virtual machine.

How to run the Docker without using the Linux?  
-> Most developers will run a Virtual Machine, Containing a full Docker installation.

# Docker With Spring Boot 3.2.0, Test Containers

## Another Udemy course: https://capgemini.udemy.com/course/docker-java/learn/lecture/32781556#overview Instructor: [Vinoth Selvaraj](https://capgemini.udemy.com/user/vinoth-selvaraj/)



-> Hardware (CPU/RAM) – we set-up a physical hardware, based on our application scale-up  
-> OS – Operating system, we can use windows, Linux, and Mac operating systems  
-> Hypervisor – It is a software which is which provides a virtualization then using this we create VMs for each and every application then we set-up the application.

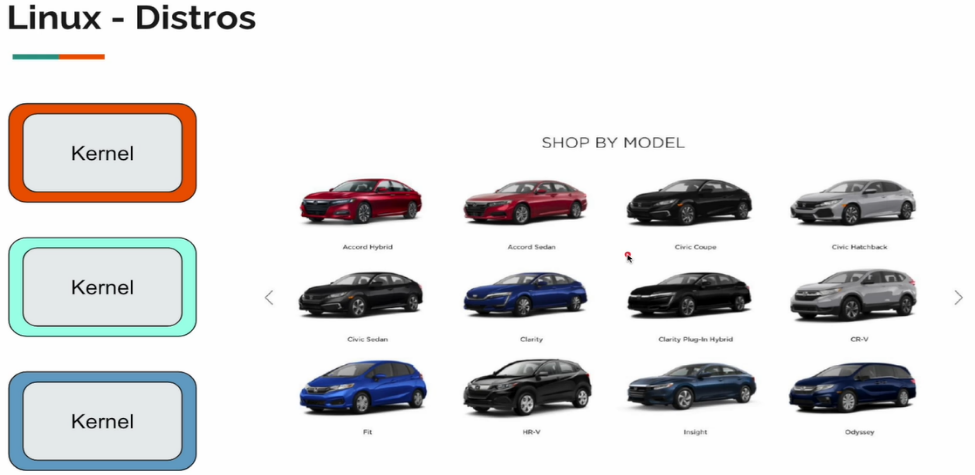
However, it will not really efficient, because here we have CPU and memory OS Hypervisor this VMs they all need, so most of the resources were used by these applications so we actual application which will we wanted to run where only using the left-over CPU and memory.

Solomon hykes introduced docker and he completely changed the way which we package and run the application.

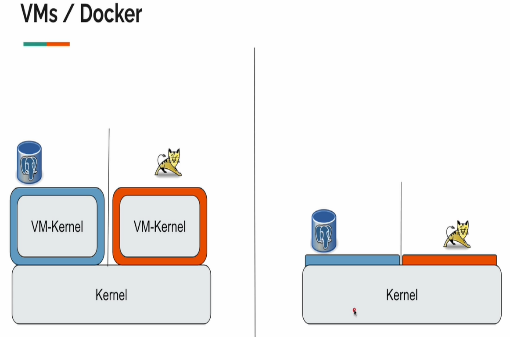
Before we are understanding the Docker, we understand **kernel** It is a heart of the operating system, when the server is starting its loaded into the memory and it acts like a bridge between the application and the hardware kernel is a main car engine, it will run but we do not have staring break and accelerator etc. So, we cannot use it easily.

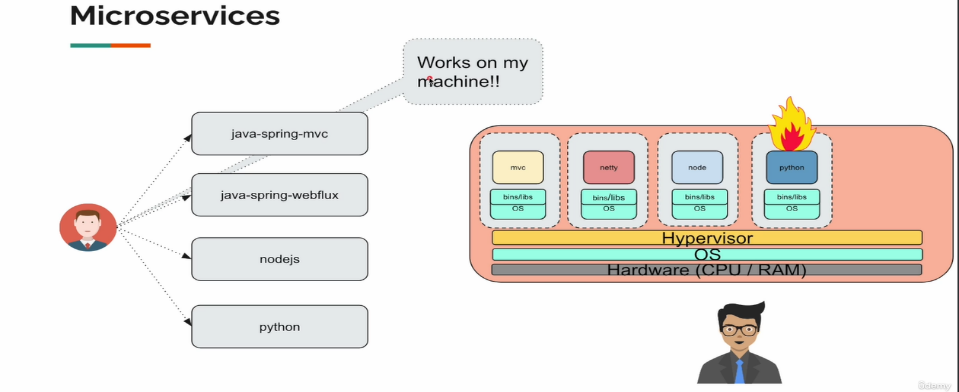


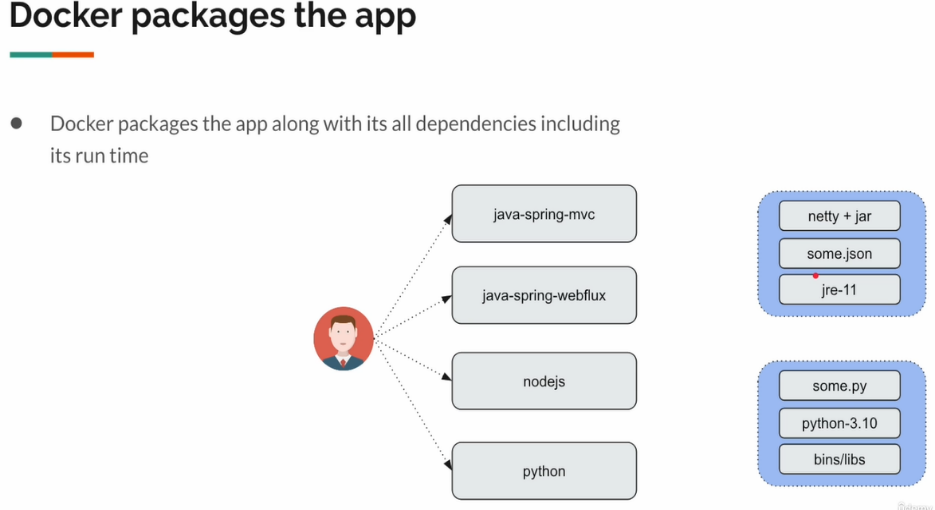
We can’t use kernel directly so, we add a layer with a set of utilities like our home directory in our Mac, Windows, Linux a browser, a terminal all this tools to make it easy to use the machine, the whole set-up then become OS.



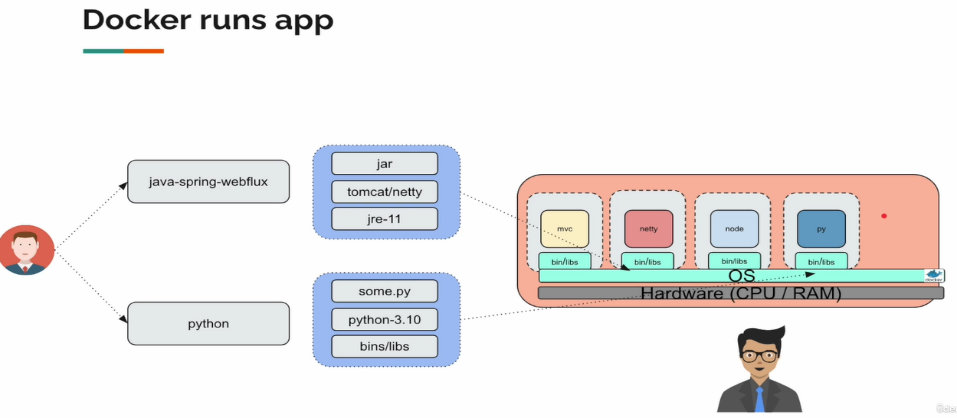
Linux server as various flavours like ubuntu, fedora, CentOS, redhat etc. Actually, all these various Linux distros they all use same kernel only the outer layer is difference among them for ex this is equalant to the various car models we have.

This is where docker is a light weight compared to the VMs.

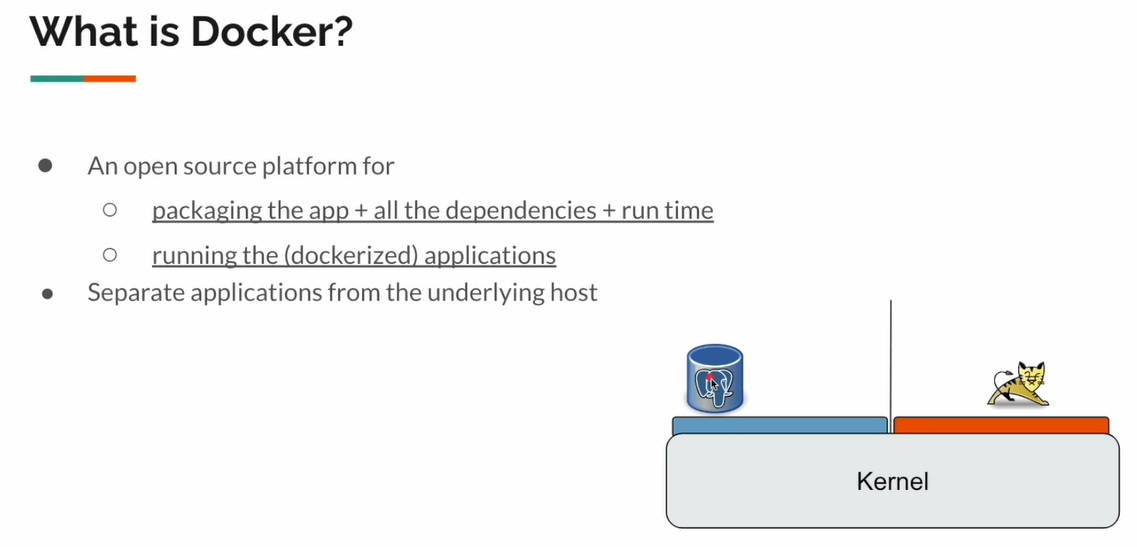
Why are we using a Docker?  
-> We as a developer develop multiple applications like spring-mvc, web flux, NodeJS and python etc. each and every application as its own set-up of dependencies and the application is works fine in our machine, I think to ready to deploy I work with the devops engineer, sre to deploy the application during the traditional virtual machine give the instructions they set-up the application, but its not working in production because of some wearied exceptions, we developer will be like its work on my machine what will be saying, so, this is the problem docker is trying to solve for us.

So instead of working with the system administrators and providing the instructions, we define the application dependencies in a simple text file. So docker read all these instructions and it will package our application along with the all-web dependencies including the runtime.

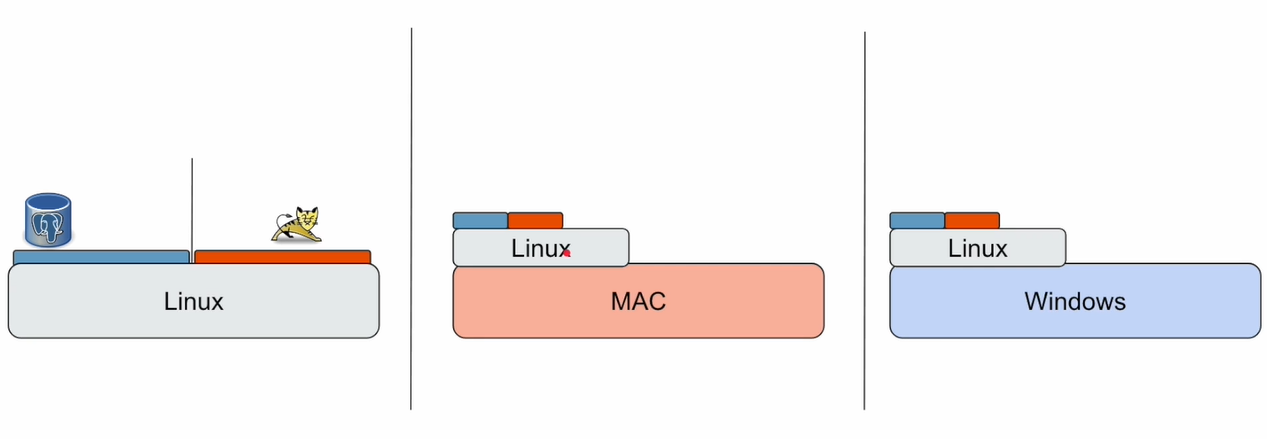
In a good old day, we are packaging the applications jar or war files in the modern cloud native application development process we take one extra step to package the application as **Docker image**.

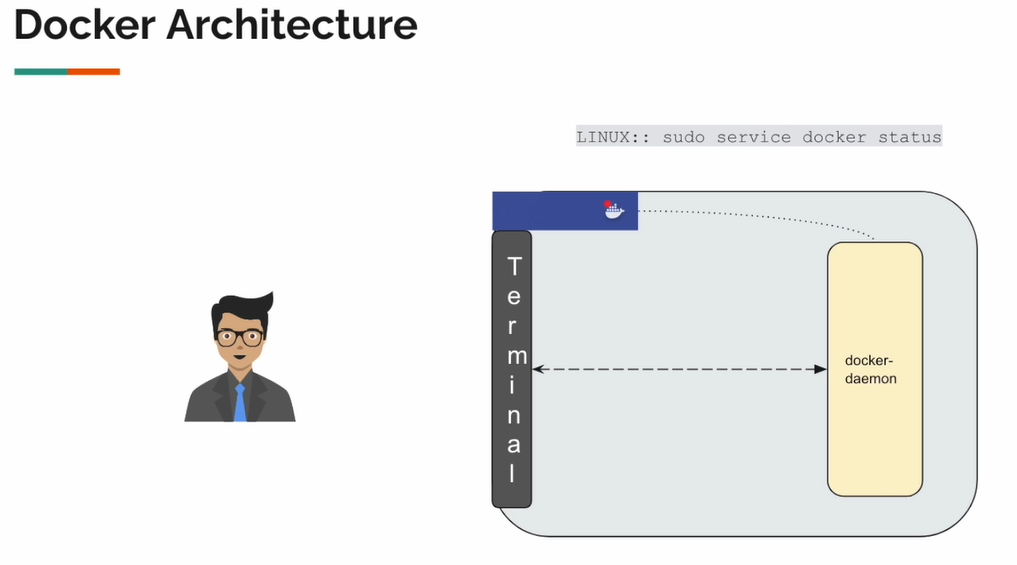


So, using docker to package the application and when I give the docker packaged application to another machine when that machine as docker installed we do not need anything else it will just run fine, we do not have to worry tomcat install, dependency version problem jre installation problem etc. we don’t have to worry about anything docker will run the app just fine.



How Docker works on Mac/Windows?

Docker cannot work on windows and mac directly, so when we install docker in these machines docker will install a virtual machine actually it needs Linux kernel to work but it will work just install only one.   


Source code:  
  
1) GitHub link for the Source Code (updated for latest spring boot 3.2 and Java 21).  
 <https://github.com/vinsguru/docker-spring-webflux>  
  
2) Docker commands we discuss in this course  
 <https://vins-udemy.s3.amazonaws.com/docker-webflux/docker-commands.pdf>  
  
3) A Linux commands cheat sheet which I had used  
 <https://vins-udemy.s3.amazonaws.com/docker-webflux/linux-cheat-sheet.pdf>  
  
****  
Once we launch the Docker we use the Commands are: