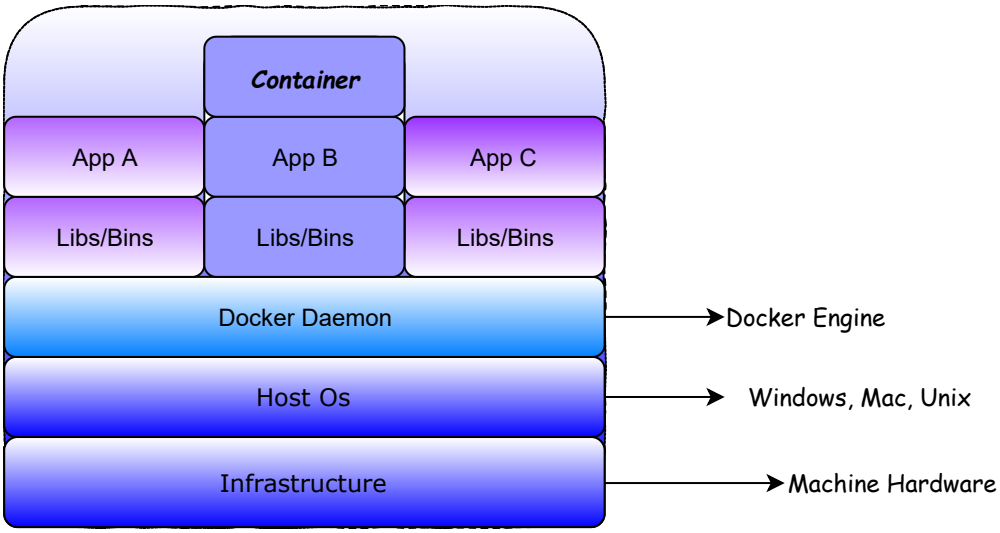
Docker Architecture - Image, Container

Let's look behind the scenes of Docker!

We'll cover the following

- Definition from Docker's creators
- Image
- Container



In the last lesson, we saw the issues our industry was facing before Docker's inception. But how did Docker manage to solve those issues and deliver a revolutionary product to the world?

To find the answer to this question, let's take a deep dive into the architecture of Docker to understand Docker provides all these functionalities.

Definition from Docker's creators

As defined by Docker's official documentation, "Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allow you to run many containers simultaneously on a given host. Containers are lightweight because they don't need the extra load of a hypervisor, but run directly within the host machine's kernel. This means you can run more containers on a given hardware combination

than if you were using virtual machines".

You may not understand this definition now as the term "*container*" is still unfamiliar. Since this "container" is powerful enough to solve many of the industry's problems, it must be the core of Docker.

So, let's learn the secret behind the container: what it is and how to create one?

Image

Before we learn about containers, we need to first understand the '**image**' term in Docker. If you look at the diagram above, the top layer of apps and libs/bins above the Docker daemon is encapsulated in the container. Basically, the app is packaged with libraries and binaries required by it. But how does Docker achieve this packaging?

Docker provides the facility to create a custom image on top of the Linux kernel with your app and its libraries. The image is a blueprint of the container and the container is created from it.

For simplicity, if you take this from an object-oriented programming point of view, an image is a class, where all the requirements are defined and declared. A container is an instance of the image. These images are stored somewhere in the cloud and pulled as needed.

Container

Container: A container is an instance of an image, which simulates the required environment with the use of the Linux kernel packaged in it. In the diagram, you can see app B is enclosed in one container. Similarly, you can enclose the other two apps as well.

You might ask, how a container provides the required environment with isolation if it shares the OS kernel. Well, that is done with the help of images.

Let's take an example here. If your app only needs Python 3.5 from the system, you will only need that in your production environment as a dependency. Everything else will be an extra overhead. So, Docker provides the template built on the Linux kernel with the needed dependencies only and nothing is installed in that template. That template is called an image.

So, if you fetch a Python 3.5 image from Docker and run an instance of it, you can do whatever you were able to do in the host machine using a command line

interface. If you make any mistake with that instance, you can delete the container and create a new one from the existing image. This way, your main environment remains intact in the form of an image and you can play around with the dependencies packaged in the image using containers.

In the next lesson, we will see how Docker manages all these using its client-server architecture.