# Containers:

- ➔ Let's first understand what problem we will face if we are not using containers while developing the application and deploying it over the server?

- ➔ Suppose, there is a developer 'A' who is working on a data science project and is assigned with a WINDOWS machine. So, this developer will need to install many dependencies on his/her windows machine, that will enable him/her to work over assigned data science project. These dependencies can be Mysql, Anaconda, mogodb, pythonic libraries etc.

- ➔ Say after few days developer B joins the data science team to work along developer A. let's assume that this developer is assigned with a linux machine. So, due to libraries mismatch or configuration changes data science application that developer A has built might not work over Developer B linux machine. This is because Developer A built application compatible to his/her WINDOWS machine which is not configured to run on different machine like LINUX. A common example can be the way pythonic libraries are installed over WINDOWS and LINUX machine are different or the Anaconda installation for WINDOWS and LINUX are different.

- ➔ Assume that Developer A helped Developer B to set up the Dev Environment with some changes and now both of them worked on their respective Dev environments and implemented certain data science stories which is then sent to QA team for Quality check.

- ➔ QA team will be having their own QA server to perform their job. Let's assume that they missed to install some dependencies due to which again application will not run and QA team will start chasing developers for this. Again Developer has to spend time and configure the application so that it can run on QA server.

- ➔ So, to conclude before containers were introduced one has to manually install all the dependencies and configurations, when application is moved to some new environment. This is the major issue as the team who is doing this setup may not have enough knowledge due to which they may miss certain libraries/dependencies due to which application may not run. This issue gave birth to CONTAINERS which is used to build containerized applications that can solve this issue.

- ➔ Containers is the way to package application with all the necessary dependencies and configurations.

- ➔ Containers helps in creating portable artifacts which can be easily shared and moved as a package to any environment.

- ➔ We can also, understand containers with one classical example.
  Say for example we are shifting from House A to House B. To make this shift happen we will place all our essentials from House A (TV, washing machine, kitchen utensils, bed, clothes etc) inside a container followed by shipping this container to House B using truck. Once, container reaches House B then we will simply UNPACK all our items from container and place it in House B. This is how exactly we are working with Containers during development

and deployment of any application. In Dev environment we are placing all the dependencies and configurations inside a container in form of base images and then move or ship them to different environments(QA, UAT, Prod etc.) and run this container over there.

➔ **Containers Vs Docker:**
Docker is an open source platform. Container is a concept that talks about containerization of applications by packaging all the necessary dependencies under the single bucket. Whereas, Docker is a vendor that provides the tool and technology to build such containerized applications and ship them across different environments.

➔ Container is divided into multiple layers which is called base images. In each base image we will be storing a dependency (E.g. anaconda, python3.5 etc).

➔ When we combine all the base images, the image thus obtained is called Docker Image. When we run Docker image then a environment is created where all dependencies represented by base images within Docker image is installed so that our application can run in the resultant environment. This resultant environment is what we call container.

➔ To conclude, we can say that on running Docker image an environment is created which is called Container.

# Docker Vs Virtual machine:

➔ Both Docker and Virtual machine does virtualization. If we want to create a virtual OS environment, this can be done with Virtual machine but still why we prefer Docker for any such task?

➔ Let's first understand about Operating system.
Operating system has 3 layers:
1. **Application layer:** Where application is stored. User interacts with this layer via Application UI.
2. **OS kernel layer:** This is the layer that establishes connection b/w Application layer and hardware layer.
3**. Hardware layer:** This is the layer that consists of hardware devices.

➔ Docker interacts with only Application layer of OS. This means that Docker does virtualization of only Application layer of OS.

➔ Whereas, VM interacts with Application layer and OS kernel. This means that VM does virtualization of Application layer and OS Kernel.

➔ Please note when we say that VM virtualizes Application layer and OS kernel then it means that VM creates a virtual application layer and virtual OS kernel by using the real resources (like CPU, RAM, ROM etc.) of host OS. Same implies on Docker as well.

- ➔ Since, Docker only virtualizes the Application layer it is usually of smaller size due to which they start and run much faster as compared to VM that virtualizes both Application layer and OS kernel. Due to this reason we prefer Docker over VM for containerizing our applications.

- ➔ Only, disadvantage of using Docker images is that it may give compatibility issues. For E.g. For below Windows 10 version docker images for LINUX based OS will give compatibility issues.
  Whereas, VM does not have compatibility issue. That is we can install VM on any OS whereas, for Docker images, if we are installing/unpacking LINUX OS based docker image on a WINDOWS based OS below version 10 then it may give compatibility issue.