If I run whatever application on my local machine, according to some permission and use cases this application can access IO hardware. (Receive data from internet, save on hard drive, send data to printer).

This is possible because between application and hardware resides:

1. OS
2. Drivers what connect hardware devices and OS

**Virtualization** software runs on an HOST OS.  
  
What is our starting point is a virtualization of the hardware.  
  
Software receives possibility to access hardware (like all ohter sw on Host OS) delegating operation on the Host OS.  
Virtual machine starting point is a machine with some hardware (configured RAM, CPU and devices) and no OS.  
TO use a VM we need to install a Gues OS and every action we perform through that on the hardware is delegated to Host OS through the VM sw.

**Containerization**

Assumes Linux OS kernel as starting point.  
Delegates to this kernel all calls for IO with hardware.  
  
We can create images with additional layer of sw on the top of the (provided) Linux OS kernel.  
  
When docker

loads an image, the layers of sw packages what are described will be saved locally for later use.  
  
runs a container, local packages (in the form of an image) are made available to the host Linux Kernel (Host OS).   
All interaction between container sw and host hardware are delegated to the kernel.

Many Containers in execution shares the same kernel.

In Non Linux system Docker is loading locally an optimzed virtual machine containing Linux Kernel   
  
- Hypervisor-V Virtual Machine (in Win 10)

How does it work in Windows?

Answer from

Docker for Windows still uses a Linux VM to run Linux containers. But instead of using Virtual Box (which is what is used with Docker Toolbox), the Linux VM is run using Hyper-V - a Windows-native hypervisor. This means that Docker for Windows ships fewer components and has less moving parts.

If you install Docker for Windows and run docker version you'll see that the Docker Linux daemon is running on "Moby Linux".

Example Backend Stack:

Our Application:  
Services: MySql, Tomcat  
Libs: (JDK)  
Os: Fedora (GNU tools and libraries, UI)  
Os: Linux (Kernel) (io hardware and package manager)

**QUOTES:**

From article:

<https://xebia.com/blog/deep-dive-into-windows-server-containers-and-docker-part-2-underlying-implementation-of-windows-server-containers/>

Difference between containers and VMs

* **Level of virtualization:**containers are a new level of virtualization. Looking at the history of virtualization it started with concepts like virtual memory and virtual machines.   
    
  Containers are the next level of this virtualization trend. Where VMs are a result of hardware virtualization, containers are a result of OS virtualization.   
    
  This means that where hardware virtualization let the VM believe that its hardware resources are dedicated to that instance, OS virtualization let the container believe that the OS instance is dedicated to that container.   
    
  This difference in virtualization is importance to notice.

Containers do for example not have their own kernel mode. For this reason containers can’t be seen as VMs and will also not been recognized as VMs within the Operating System (you can try it yourself with the PowerShell Get-VM commandlet). A good analogy to explain this difference is that of houses (VM’s) and apartment buildings (containers). Houses (the VMs) are fully self-contained and offer protection from unwanted guests. They also each possess their own infrastructure – plumbing, heating, electrical, etc. Apartments (the containers) also offer protection from unwanted guests, but they are built around shared infrastructure. The apartment building (Docker Host) shares plumbing, heating, electrical, etc. While they both may share some characteristics it are different entities.

* **Dealing with OS:**another important difference between containers and VMs is the way how both artifacts deal with the kernel mode.   
    
  Where VMs have a full OS (and dedicated kernel mode) available, containers are sharing the “OS (actually the kernel mode)” with other containers and the container host.   
    
  As a result containers should align with the OS of the container host while VMs can pick the OS (version and type) they like.   
  Where VMs are able to run a Linux OS on top of a Windows hypervisor, with the container technology it is not possible to run a Linux container on a Windows container host and vice versa.
* **Growth model:**containers share the underlying resources of the container host and build an image that is exactly what you need to run your application. You start with the basics and you add what you need. VMs are built in the opposite direction. Most of the time we start with a full operating system and, depending on the application, strip out the things we don’t want.