Containers & Kubernetes

In the second lecture we have talked about how applications interact with the kernel and the hardware. By now you know that applications running in ring 3 do not have direct access to things like the disk. To access those details they need to execute a system call to the kernel. The kernel will then give the application the details required, for example a file from the disk.

Let's play a hypothetical game: process A wants to access a file called `test.txt`. It calls the kernel, the file is opened, and process A is now free to read the contents. When process B comes along and does the same, it will receive the same file.

Or does it? In the Linux kernel the filesystem root directory (starting with `/`) is a virtual construct that can be changed on a per-process basis. It is possible to make process A see a completely different folder structure then process B. This mechanism is called chroot. It is, in fact, so old that it predates Linux by more than a decade and was first present in Version 7 Unix in 1979.

> **Try it out yourself!**
>
> Run the following command on a 64 bit x86 Linux in an empty folder:
>
> ```
> mkdir -p ./bin &&\
> curl "https://www.busybox.net/downloads/binaries/1.21.1/busybox-x86_64" -o ./bin/busybox && \
> chmod +x ./bin/busybox && \
> ./bin/busybox --install ./bin && \
> sudo chroot . /bin/sh
> ```
>
> You should now be able to run `ls -la` or similar commands and see that you are in an almost empty filesystem with no ability to access the files outside.

This alone only isolates the two process on a filesystem level, there are still plenty of opportunities for two processes to interact, for example:

- On the network
- Interprocess Communication (IPC)
- Sending signals to each other by process ID
- etc.

In the early 2000's there were two projects that attempted to implement process isolation in the Linux kernel: OpenVZ and Linux-VServer. Running these projects required patching and recompiling the Linux kernel from source.

Starting around 2006 several companies, including Google, IBM, SGI, Bull, and the OpenVZ project itself has put significant effort into taking OpenVZ apart and piece by piece and submitting it to the Linux kernel. The User Beancounters from OpenVZ, for example, became cgroups and allow resource allocation to a group of processes.

> **Do you want to know more about history?**

> The history of containerization is surprisingly hard to piece together despite the relative young age of the technology. If you want a bit of a deeper dive take a look at “A Brief History of Containers (by Jeff Victor & Kir Kolyshkin)”.

All these isolation technologies, together, form what's today known as Linux containers. Windows has undergone a similar development in recent years, and has *virtualized* the typical Windows interaction points between processe: job objects, process table, registry, filesystem, etc.

In essence, *containers don't exist*. They are a collection of technologies that provide process-level isolation for an application.