Article
Luca Ravazzolo · Jan 31, 2018 3m read

Container - What is a Container?

Containers

With the launch of InterSystems IRIS Data Platform, we provide our product even in a Docker container. But what is a container?

The fundamental container definition is that of a sandbox for a process.

Containers are software-defined packages that have some similarities to virtual machines (VM) like for example they can be executed.

Containers provide isolation without a full OS emulation. Containers are therefore much lighter than a VM.

In their essence, containers are an answer to the issue of how to reliably move an application from a system to another and guarantee that it will work. By encapsulating all application dependencies inside a container and creating a process isolation space, there is a higher degree of guarantee that the application solution will run when moved between platforms.

An operating system allows us to run processes. These processes share address spaces, namespaces, cgroup, etc., and in general, they have access to the whole OS environment and the OS schedules and manages them. All that is a good thing, however, what if we wanted to isolate a particular process or number of processes to run a specific task, operation or service? In short, that capability to isolate a process is what containers offer us. Hence we could define a container as a sandbox for a process.

What is a sandbox, then? It is an isolation level within which a container has its process. This feature is implemented via the Linux kernel feature called namespaces (https://en.wikipedia.org/wiki/Linux_namespaces) which also allows for other important system parts to be sandboxed like network interfaces, mount points, interprocess communications (IPC) and universal time-sharing (uts).

The container or sandbox can also be governed or controlled via another kernel feature called control groups or cgroups (https://en.wikipedia.org/wiki/Cgroups). The rules we give the containers are used to make sure the container can be a good neighbor in sharing the resources with other containers and the host.

To understand how a container differs from a VM, we could use the analogy that a VM is like a *house* while a container is like an *apartment*.

VMs are self-contained and independent like a single standing house. Each house has its infrastructure: plumbing, heating, electrical, etc. A house also has minimum requirements (at least 1 bedroom, 1 roof, etc.).

Containers instead are built to leverage a shared infrastructure so we can compare them to an apartment. The apartment building shares the plumbing, heating, electrical system, main entrance, lifts, etc. In the same way, containers leverage the available resources of the host via the Linux kernel. Also, consider that apartments come in different sizes and shapes.

Because containers do not have a full OS but only the minimum required Linux OS needed, like some executables in /bin, some configuration and definition files in /etc and few other files, they can be very small in size which makes them very nimble when it's time to move them around or spin them up in 1 second flat. That translates into agility from the moment one builds them, throughout the provisioning pipeline of the software factory and all the way to the final run in production. Incidentally, containers fit like gloves in a CI/CD microservices architecture context but that's another story.

The processes in the container are tightly coupled with the lifecycle of the container. When I *start* a container I typically want all the services of my app to be up and running (as an example, think of port 80 for one web server container and of port 57772 and 1972 for an InterSystems IRIS container). When I *stop* the container all the processes will be stopped too.

What I have described in this post is the fundamental notion of the runtime of a container, it's sandbox that isolates its processes from the host and other containers.

There is another part to understanding containers that is about their images. That will be covered in a second post.

#Best Practices #Cloud #Containerization #Docker #System Administration #InterSystems IRIS

Source URL: https://community.intersystems.com/post/container-what-container