Container, Hypervisor and Realtime

The embedded world is currently undergoing a great upheaval. Until a few years ago, it was still common practice to develop the software for a device as a large, proprietary monolith and to avoid touching it after its introduction to the market. Thus, the device itself was also functionally a monolith, which at most exchanged data with others via a proprietary protocol.

Today, completely different paradigms have to be observed. IoT (or whatever you may call it) has opened the doors, communication is at the top of the priority list, development is accelerating and industry has recognized the value of open source software as a means of finding solutions in this race.

In addition, hardware is becoming increasingly powerful while costs are falling (the only industry where this happens, by the way). This means that multi-core CPUs, for example, are now also ac-ceptable for the embedded world in terms of the BOM. And so are large memories, standardized communication interfaces (Ethernet) and whatever else exists in the IT world. However, this new, powerful hardware can only be used quickly and cost-efficiently with open source software. It is no coincidence that Linux has established itself so quickly in the industrial environment. There is hardly any other operating system that supports such a large number of hardware platforms.

In addition to these changes to hardware and software, the role of automation has also changed in the course of cyber-physical systems. And while at the beginning everyone thought that the only solution was large, server-centric approaches (cloud), today it becomes obvious that a viable way is to bring parts of the IT world into the OT world. These approaches, known as edge and fog computing, provide for real-time data (= data with low latency, like control or sensor data) to be processed on site. A type of data collection point, which is the Edge or Fog computer, is then located above the controller. There it receives the (possibly pre-processed) data from the sensors/controllers, processes the real-time information and then transmits the aggregated information to the server farms in the cloud. Due to the advances made by hardware manufacturers, Edge computers have now the capacity to run applications that previously were running on servers on site. Consequently the programming approaches of the IT world are entering the embedded world. Now they have to deal with virtual machines (VM) and their management software (Virtual Machine Manager, VMM, also called hypervisor) or containers and assess what is likely to be the better solution. However, specific embedded requirements such as real-time must still be taken into account and fulfilled.

But what actually is a virtual machine and what is a container?

The issue of virtualization was already discussed at a surprisingly early stage: It all started more than 50 years ago, when IBM stepped into the first research on virtual machines on an IBM 360 mainframe computer. In order to manage the virtual machines on the 360, the Virtual Machine Manager, also called hypervisor, was introduced. The virtual machines pretended to the applica-tion and the operating system they were pieces of hardware, but they weren't. In fact they were generated by a software on the mainframe only virtually.

Popek and Goldberg[1] presented a study in 1974, its classification of a hypervisor is still valid to-day.

In virtualization, all resources required by a system are provided by the hypervisor. Non-existent hardware or even just access to a hardware from the virtual machine is simulated by software. This process is called emulation. A process that consumes resources and is far less powerful than real hardware.

To avoid this loss of performance, paravirtualization has been introduced. The operating system is not allowed to presume that it had all the hardware for itself. In practice, this means that there is a software interface in the hypervisor that is able to allow the guest system direct (physical) access to the resources of the physical system. The guest system must have its own drive rs, which communicate with the hypervisor and skip the emulator.

A virtual machine always is a package of operating system and application code that runs in the virtual environment, i.e. in user space. Using an emulator you can even pretend hardware architecture different from what is physically available, to the package.

In contrast, the container approach needs no virtual machine, nor the pretending of having its own hardware or its own operating system. The operating system is identical for all containers, all containers share the same hardware. The individual containers are separated using special operating system functions (under Linux, for example, CGROUPS and namespaces). This keeps the containers slim, which means that they do not require much additional code for their real task. Within one container there is the application (in many cases only a single, small application) with its required libs and

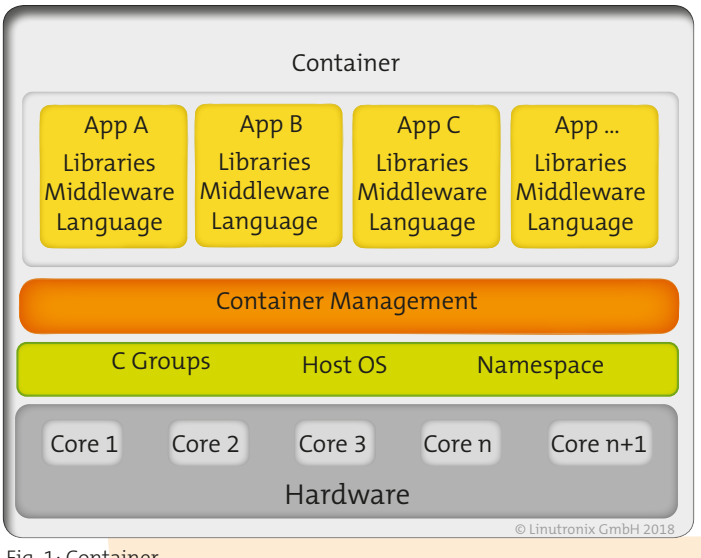

Fig. 1: Container

frameworks. The use of containers decouples an application from the infrastructure and thus offers a portability that is expected today in terms of cloud-centred approaches.

The actual application is divided into many small applications, each gets its own container, and together these so-called microservices provide the desired solution. This is supporting the DevOps approach, the creation and use of containers is simplified by standardization such as Docker and the interaction of different containers can be automated using orchestration frameworks such as Kubernetes.

IT and OR

Containers and VM were created in the IT world to solve the requirements there. By using VM it became possible to run more than one application per computer/server. And as an application in a single, large monolithic block turned into a collection of small applications, the use and utility of container technology increased.

The question now is whether these technologies can also be used in the embedded environment with its special requirements and if so, whether they can offer an advantage. Let's start with the VMs:

Considering that virtual machines were born on mainframes and then grew up on special server architectures, it can be doubted whether they fit into the OR area. However, the major CPU manufacturers such as Intel, AMD and ARM have provided birth assistance for the embedded sector: They started to extend their processors with special functions for virtualization around 2005. The technology has now entered the low-cost CPUs that are applied in the OR area.

By combining these new CPUs with a suitable hypervisor the specific requirements of automation can be covered, too. One of the most important technical requirements for a control system is real-time capability. From an economical, cost saving point of view, you would like to see as much functionality as possible on one computer. Consolidation is the resulting requirement, i.e. the combining of previously separate functions of several hardware components in just one hardware. Please, just do so, but retain the software that already exists.

A hypervisor like jailhouse is the absolutely right approach for this. An open source solution runs on different architectures, has a minimum footprint of < 10 000 lines of code and allows the use of different operating systems as well as bare metal code in the guest system. Jailhouse allows applications to be separated without losing real-time capability. The additional overhead caused by jail-house in an RT system is in the range of 2 - 10 µsec, depending on the hardware used. This allows hard real-time requirements to be met even in the guest system.

The separation allows, for example, both the graphical control unit and a certified code to run on the same CPU but in different VMs. While the GUI and its underlying operating

system communicate with the world and, for security reasons, regularly receive an update, the certified application remains in its VM, without update and without external influence. And the time-critical control runs as a bare metal application on a different core. A classical case of consolidation. The embedded world is ready for virtualization!

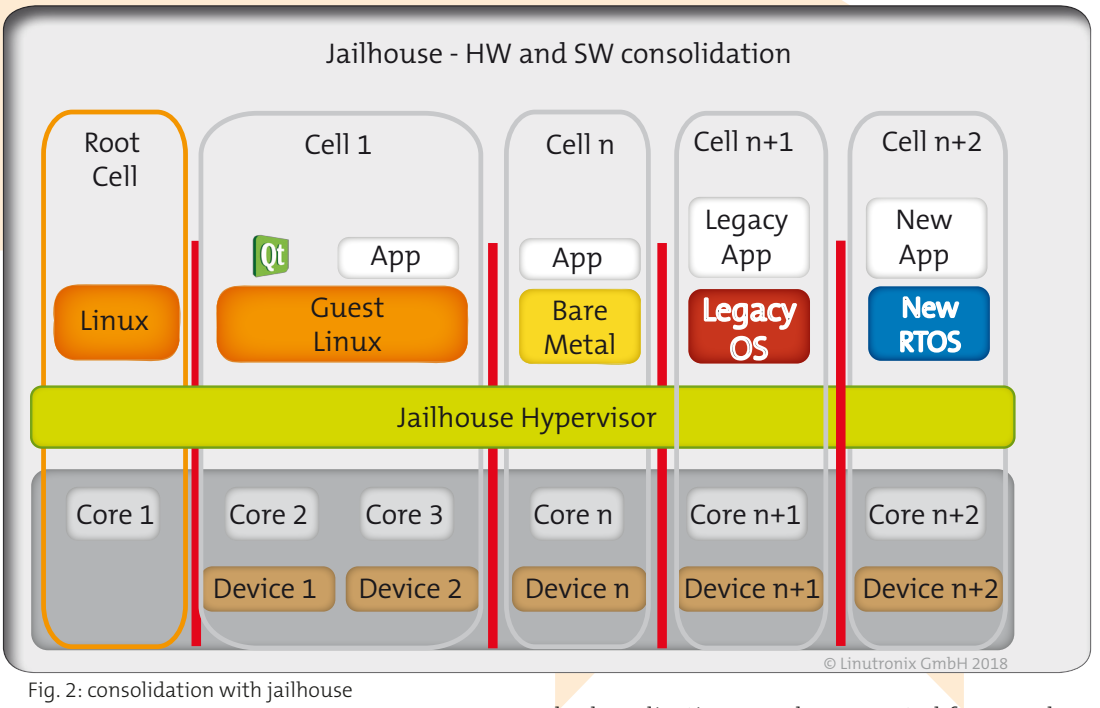

Fig. 2: consolidation with jailhouse

And what about the container approach? When Docker, an open source platform for container management, saw the light of day around 2014, a hype began, the end of which is not yet in sight. And which has also reached Embedded World. Two changing developments have complemented each other here - the triumphant advance of (real-time) Linux and the container as an (apparent) answer to the question of how applications can simply be brought to the fragmented world of Linux on as large a number of distributions as possible. In the course of IIoT (also called Industry 4.0), the use of Open Source solutions has been accepted in various areas.

Even manufacturers of closed control solutions (keyword: PLC) had to learn in the meantime that third party software must be allowed to run on their computers. Otherwise there is a risk of customer migration. So Linux is getting employed as the operation system, The PLC runs as a soft PLC (as before) and the „foreign“ software runs in the container, secured against the controller. A symbiotic approach that leaves happy participants behind and makes the previously closed solutions fit for industry 4.0 and its requirements.

There is still the question of what is the „right“ container. And that‘s not easy to answer. Everyone is talking about Docker, but also the implicit Linux tools like LXC are competing, and then there are complete distributions like Snap-OS, which jump on this topic. There is not the one answer to this question. What these approaches have in common is, for example, their real-time capability. This feature is primarily a consequence of the operating system used, but is also available to any func-tion in a container without restriction. And the individual applications can be separated from each other to a certain extent. However, it is still possible to influence the operating system. A certified application software can therefore be excluded from the update, since it is constantly being installed in a container. An update of the OS, however, could affect the application.

So what is the right solution?

There is no clear answer to this question. It depends, as is quite normal in life, on the circumstances. Let‘s take a look at the differences between the various approaches.

Container versus VM

Docker is a container-based technology, and containers represent only the user space of the oper-ating system. Under the hood???, containers are only processes that are isolated from the rest of the system. They are run by a specific image that contains all the files needed to support the pro-cesses.

Docker is designed to run applications while containers running in Docker are sharing the host OS kernel. In contrast, virtual machines are not based on containers, but are built from the user space plus kernel space of an operating system. Under VMs, the server hardware is virtualized. Each VM has an operating system and applications, and it shares hardware resources from the host.

Both VMs and dockers have their specific advantages and disadvantages. In a VM environment, each workload requires a complete OS – in a container environment, multiple workloads run in one OS. The larger the OS footprint, the better container environments pay off (as far as only resource consumption is taken into account). In addition, containers offer additional benefits such as reduced IT management costs, smaller snapshots, faster application startups, reduced and simplified security updates, and less code for transferring, migrating and loading workloads. Simplified security update means that only the underlying OS needs its patches.

The containers are more or less shielded from each other. But by far not as separated and thus secured as a guest OS in a hypervisor system. And their data integrity is largely based on the OS used.

Both, a guest OS as well as the containers, allow third party programs to be executed without compromising the security of the entire system. In addition to the actual application, a container only contains those components that are still needed by the application software, such as libraries, and so on. And a container can easily be supplemented by further functionalities such as debug tools, etc. So, for example, you can deliver a significantly extended range of functions for the test phase compared to the subsequent production phase. This can be done without great effort on the part of the creator. A virtualized guest OS can do the same, but the application must always be exactly matched to the guest OS and its components, or vice versa. To put it simply, the guest OS plus application makes more software than a container.

In the IT world, the portability and scalability of applications is of crucial importance. Here, for example, applications are scaled across many servers. Containers support this scalability by design. This dynamic scalability is not part of the OT world. There a certain number of resources are available, which does not change dynamically according to the load. Even with on-premise clouds (also called edge or fog solutions) the application is limited to the local resources. That advantage of container technology is not required in the OT world.

Systems with certified software need to be looked at separately. We are not talking here about safety-certified systems. Rather about systems which, for example, measure a specific physical quantity and are approved by a certification body such as PTB (Physikalisch-technische Bundesanstalt Braunschweig). If such a system is connected to the Internet in the course of Industry 4.0, for example, the topic of updates becomes urgent the same moment. At any rate the system adopted must not be changed. But how can you update the rest of the system without compromising certification? With a little goodwill on the part of the certification body and an accordingly additional software effort, this may be possible with containers (the author knows of appropriate solutions). However, the solution is simpler and cleaner with one virtual machine, whereby the measuring software runs in its own VM and the „rest“, i.e. the communication part, in another. The architecture itself ensures that an update only affects those non-certified system parts which may or need to be changed.

Whether the plus of software and data integrity on the VM side outweighs the leaner software volume and the reduced data integrity on the container side or not, must be decided in the context of the application (e.g. consolidation of legacy code) and the customer requirements (e.g. certification or maintainability). Anyway, it is an obvious fact that new IT technologies are being used more and more quickly even in the OT world. The OT area can no longer afford its isolated solutions like in the past, because customers would no longer accept and neither pay for it.

Are you interested? Would you like to learn more about our products and solutions? Simply contact us via telephone or email.

01/2019_V1.1

LINUTRONIX GMBH
Bahnhofstr. 3 | D-88690 Uhldingen - Mühlhofen
Telefon +49 7556 25 999 0 | Fax +49 7556 25 999 99
sales@linutronix.de | www.linutronix.de

LINUTRONIX
LINUX FOR INDUSTRY