Mission CRITICAL

Containerization Approaches for the IIoT

New application gateways enable low-power edge intelligence

April 30, 2020

Tim Winter

Containerization approaches bring advantages to the operation and maintenance of systems across physical compute resources. In the enterprise IT world, containers are leveraged to decouple computational workloads from the computing substrate on which they run. This allows, for example, compute hardware to be treated more as a utility, allowing deployment of multiple workloads across racks, scaling the hardware resources, such as processors, memory, and storage, as necessary to handle the workloads.

Multiplexing software loads across fixed hardware resources allows for more efficient use of the hardware investment and more robustness against hardware faults. It also enables easier maintenance and evolution of the software workloads themselves by allowing schemes where a centralized container provisioning or configuration can be updated and then pushed out to the execution environment. Containerization technologies as applied to traditional enterprise IT have been a key enabler of the modern cloud.

Virtual Machines

Typically, a container can be thought of as a lightweight virtual machine. A full virtual machine is capable of complete emulation of a target hardware layer on a host machine, including the CPU instruction set, peripheral set, etc. Virtual machines are capable of offering high portability but incurring significant overhead due to simulating every aspect of the target machine within the host machine. This practically requires the host machine to be overspecified as compared to the target machine being emulated. In many cases, such level of emulation is not necessary.

Hypervisor-based virtualization requires fewer host resources than a full virtual machine. A hypervisor provides each execution environment a private view of the underlying hardware but is most often bound to the underlying host machine architecture, so it does gain some additional efficiency by constraining the hardware architecture to that of the host machine. In Industrial Internet of Things (IIoT) applications, the level of abstraction and isolation provided by full virtual machines or hypervisors is often not necessary.

Containers are not full virtual machines but, instead, operate under the constraints and architecture of the host machine. As such, containers are able to interface to the CPU architecture and low-level operating system (kernel) of a host machine, directly sharing the hardware and kernel resources of the host machine.

Containers depend on the low-level operating system of the host machin, but can encapsulate and provide portions of the higher layer operating system (userspace). This allows an application within the container to be built and run against a private, fixed set of versioned operating system resources.

Partitioning

Most system administrators or UNIX application developers are probably familiar with the concept of “dependency hell” — making all of the system resources available in order for an application to run. It can often be a tricky and tedious exercise to maintain multiple application dependencies across all applications that are provisioned to run on the same server. Containers allow each application to bundle a controlled set of dependencies with the application so that these applications can independently have stable execution environments, partitioned and isolated from other containerized applications on the same server. Even application updates are often packaged and deployed as container updates for convenience. Thus, containers provide strong partitioning between application components on a target machine.

Enhanced Security

Since containers execute within the context of a container engine, it allows enhanced security policies and constraints to be imposed on an application by constraining the container engine itself. In a Linux hosted environment, for example, using mechanisms like 'cgroups,' process space isolation, file system controls, and kernel-level mandatory access controls, the container engine can be forcibly constrained to operate under those controls — e.g., to limit memory usage, CPU usage, access to specific parts of a file system, access to network resources, or to allow only certain a priori-approved subsets of kernel operations.

By applying those constraints through the mechanism of the container engine, such security controls are imposed, even if the enclosed application is unaware or uncooperative to participate in those controls. This is consistent with modern IT security best practices.

Containers, similar to applications, can be signed and authenticated such that the content is distributed to a compute node and can be authenticated and validated under strong cryptography by the container engine.

Orchestration Systems

Modern containerization systems also include or interoperate with orchestration systems. Orchestration systems provide the means to dispatch containers to host machines and to determine which containers are to be dispatched to which hosts. Additionally, most

orchestration systems allow applying configurations to parameterize containers and support for management metrics/dashboards to monitor a system. When it comes to coordinating the deployment, provisioning, and operation of containers at scale, the capabilities provided by orchestration systems are necessary.

Containerization Approaches and Benefits

In terms of constructing and maintaining containers, some systems have more capabilities and features than others. A container can always be constructed by hand, but there are often tools and materials within an open source ecosystem that can aid the effort. A modern system will usually allow a container to be derived from a composition/library of reference containers. These libraries promote reuse, leverage the ecosystem, and allow for rapid development and deployment of a container.

Broadly, containerization schemes decouple the challenge of provisioning an application and its execution environment in a controlled manner to effectively utilize underlying hardware compute resources. Containers bring benefits of partitioning, security, and orchestration. The approach is cheaper than full virtual machines and still results in duplication of operating system/user space components.

Leveraging Containerization Approaches for the IIoT

Although containerization technologies have been primarily developed for traditional enterprise IT, there are clear parallels and advantages to adopt similar schemes for the IIoT.

One thing to consider is the type of IIoT host machine on which the container shall be deployed, which often entails use case, future-proofing, and ROI considerations. In some cases, this may be a high-value installation warranting highly capable compute resources at the edge node, similar to the servers deployed in an enterprise data center. In other cases, the requirements may justify a lower-cost and lesser-capable machine to be allocated at that edge node. In a fully instrumented IIoT deployment, there will likely be different tiers of assets that are associated with different classes of edge hardware. How to economically enable each class of assets at the associated scale can quickly become an important driver in the selection of edge node hardware and architecture.

Another thing to consider is how to leverage the partitioning properties of a container, i.e., sandboxing. Is there a single, monolithic container deployed at the edge that contains all of the application functionality? Or is it preferred to get a better and more robust posture by isolating application components into separate spaces/separate containers?

For example, by partitioning edge functionalities amongst different containers, one container might be granted more privileges. An application component whose job is to periodically read, assess, and report alarms could be granted read-only privileges to interact with an edge asset. An application intended to perform a software upgrade on the edge asset would need more privileges, but different role-based security can be applied to interact with that application.

This architecture can map into a layered security approach where strong enforcement of permissions and mapping to roles can be orthogonally constrained around separate applications hosted on the same edge node. Further, being able to separate application components can lead to more robust implementations where the behavior (or misbehavior) of one application does not directly influence another. This approach also allows to easily add incremental enhancements to the edge device.

Interaction of application components with each other is an additional consideration. Since the applications are separated, an inter-process communication (IPC) scheme/remote procedure call (RPC) scheme would need to be implemented for separate applications to interact within the edge node. Such IPC/RPC schemes should also be authenticated and controlled to allow only approved interactions. Note that typical containerization schemes do not provide these mechanisms out of the box.

Security features of containerization schemes are consistent with modern operating system designs and modern security best practices. Being able to impose OS level controls and policies provides better ability to limit by design the potential impact of security breaches on a system. The mechanisms to validate and authenticate the application components that run at the edge are also consistent with the approach required by modern security posture.

Orchestration schemes have clear value in the IIoT. It is absolutely necessary to leverage a scheme for managing a fleet of IIoT edge nodes in a controlled and centralized manner to manage, version, maintain, and push containerized application components to the edge.

Unlike in a traditional IT environment, one challenge here is to group and coordinate the containers targeted to specific edge devices. Container workloads must be mapped to concrete, physical deployments of edge devices, since those devices are directly tied to field assets. The orchestration system cannot select any hardware to run a container, but needs to be flexible enough to target specific edge nodes with ease.

Orchestration schemes may not also be entirely sufficient to manage an IIoT system, as there are additional considerations of the host system that need to be managed and/or provisioned (network interfaces, VPNs, security credentials, cellular modems, etc). These resources are usually managed directly by the host operating system and simply made available for use by the container. Traditional approaches to IIoT platforms encapsulate this function under device management, where the management of containers/applications hosted in the device may be a subset of a unified device management.

Selection of an open-source or closed-source container engine also needs consideration, as there might be dependencies on third parties to maintain it. Ongoing support for third-party technologies, customizing applications within a container, evolving capabilities, and integrating with different protocol stacks and clouds are some of the other factors to consider.

Tim Winter is chief technology officer at Machfu.