Smart industry

Edge insights courtesy of containerization

Utilizing these technologies to enable low-power edge intelligence.

July 29, 2020

By Tim Winter, Machfu chief technology officer

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, computer hardware to be treated more as a utility, enabling deployment of multiple workloads across racks and scaling the hardware resources (processors, memory, 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.

Partitioning

Most system administrators or UNIX-application developers are probably familiar with the concept of dependency hell–making available all of the system resources in order for an application to run, then coordinating the same as different applications are updated on a server machine. 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, they enable 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, filesystem 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.

Orchestration systems

Modern containerization systems also include or interoperate with orchestration systems, which 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 & benefits

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, but still may result in duplication of operating system/userspace components and incur the associated overhead of additional disk space, memory, and (to a lesser extent) CPU. Some well-known containerization schemes used in enterprise IT include Containerd, Docker and Snaps. Well-known orchestration systems include Docker Swarm and Kubernetes.

Containerization approaches for the IIoT

Although containerization technologies have been primarily developed for traditional enterprise IT, there are clear parallels and advantages to adopting similar schemes for the IIoT. Likewise, there are some unique differences for IIoT to consider.

A first consideration 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.

A second consideration is using 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 greater and differing privileges as per its

function. 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. A different application, for example, capable of loading 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 you to easily add incremental enhancements to the edge device.

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 in IIoT 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, ability to customize applications within a container, evolution of capabilities, and flexibility to integrate with different protocol stacks and clouds are some of the other factors to consider.

Advantages of Android

Machfu's IIoT edge gateway leverages benefits of containerization technologies via a customized port of the Android operating system. The Android operating system has been developed for over a decade and is primarily based on an open-source platform curated by Google. It is today estimated to run on some 2.5 billion devices, across many different hardware architectures and form factors. Android provides an exhaustive and stable set of APIs allowing the implementation and deployment of customized applications.

Android runs a customized userspace on top of a Linux kernel and is able to apply the same controls using the same underlying technology that traditional containerization solutions utilize. But instead of a container that drives to the kernel interface, Android brings the container to a consistent set of platform APIs and thus provides a lightweight application container, which reduces the necessity and overhead of each container needing to enclose operating-system components.

Android also provides additional layers of permissions/controls on OS-provided services that run at a layer above the kernel–for example, accessing GPS location or to engage in cryptographic operations utilizing secure credentials. There is already a secure method for applications to provide and invoke APIs to interact with other applications, including security and access-control constraints. This enables a different level of partitioning beyond what is accommodated by a typical containerization scheme.

The necessary underlying mechanisms are built-in to the Android architecture, allowing application developers to focus on developing and customizing their own business logic. These applications operate under similar partitioning and security controls that traditional containerization strategies provide. Similar to

other containerization schemes, strong cryptographic mechanisms are used here as well. These solutions include not only the orchestration/deployment of lightweight-application containerized components, but also integrate the provisioning and management of the host system.