Deploying Embedded Applications Faster with Containers

By Rob Woolley, Principal Technologist, Wind River

WNDRVR

INTRODUCTION

Container technology is fundamentally changing how systems are being developed, tested, deployed, and managed. People are most familiar with containers as part of cloud-native architectures in which applications are decoupled from the infrastructure — including hardware and operating systems — on which they are running.

The benefits of this approach include being able to automate the software pipeline to remove manual errors, standardize tools, and accelerate the rate of product iterations. With a CI/CD pipeline, companies can leverage continuous integration (CI) where code changes are merged in a central repository with continuous delivery (CD), thereby providing the ability to automate the entire software delivery process and deliver high-quality software faster.

Embedded developers can also benefit from the infrastructure-agnostic, scalable execution environment enabled by containers. Imagine a design process — from development to test to deployment to production to management — in which developers can share resources, pipelines, and results across the team. Instead of being limited by the number of development boards available, companies could exploit the elasticity of the cloud to set up multiple instances of a system on demand.

However, traditional embedded application development and deployment has significant differences compared to a cloud-native architecture:

- It is tightly coupled to specific hardware.
- It is written in lower-level languages such as C/C++.
- It interacts directly with hardware (e.g., peripherals).
- It requires specialized development and management tools.
- It tends to have a long lifecycle and stateful execution.
- It faces an increasing diversity of end hardware and software deployed in the field.

To bridge container technology to the embedded world requires that embedded development adapt to a cloud native–inspired workflow, but in a way that maintains the requirements of applications, including real-time determinism, optimized memory footprint, an integrated tool chain for cross-compiling and linking, tools for security scanning and quality assurance, and the ability to secure the build environment.

This article will explore the use of containers in the embedded design process and address how to meet the specific performance and cybersecurity challenges particular to embedded devices operating at the edge.

TABLE OF CONTENTS

CONTAINER USE CASES

Figure 1 shows the U.S. Department of Defense's DevSecOps development pipeline and how designs progress through the pipeline. Figure 2 shows the pipeline when containers are in use. This demonstrates how tools can be integrated into workflows and shared across different groups to ensure consistency, regardless of the service they perform.

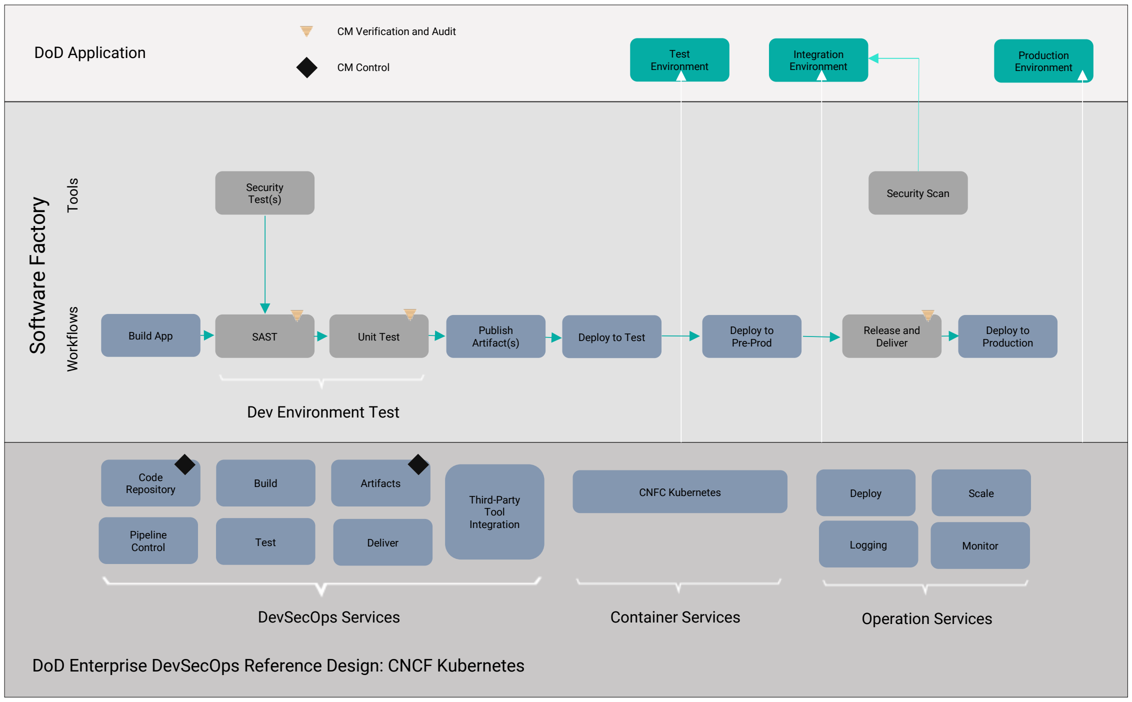

Figure 1. The U.S. Department of Defense's DevSecOps development pipeline

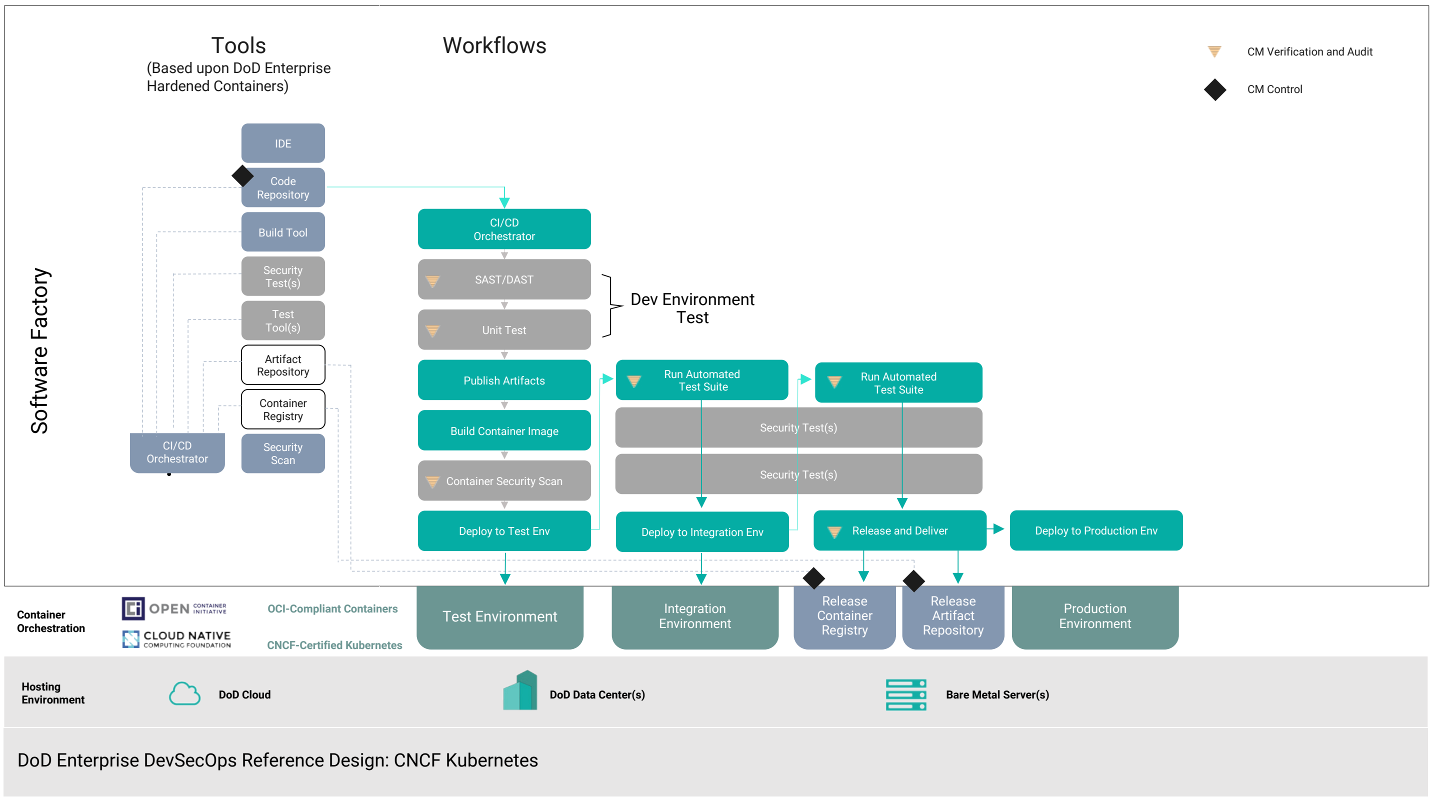

Figure 2. The same DevSecOps development pipeline when using containers

WNDRVR

Even though the design is moving through different environments, a container ensures that there is consistency between builds. For example, because tools are part of the container, groups don't need to install multiple versions of tools on their computers to work with each design. In fact, containers enable developers to revert to previous builds and produce the same results as the original build, even years after a product has been deployed.

Note that containers are not limited to development alone. There are a huge number of operational use cases that help drive the convergence of IT and OT. The same tools can be used for deployment and managing software, thus standardizing software development. Management and orchestration policies can be shared, simplifying the coordination of updates. This is especially important as the frequency of updates increases from the introduction of new capabilities, bug fixes, and necessary security updates. Containers also simplify the software supply chain. Effectively, they are a complete bill of materials, showing everything that went into building a specific product.

THE EMBEDDED DEVELOPMENT PIPELINE

As stated earlier, the requirements of embedded systems introduce a lot of new issues compared to a typical cloud-native application. For example, edge devices might be physically accessible. They may need to operate in constrained, unreliable networks. Devices may go offline for long periods of time to conserve battery life. It can be difficult to determine whether a device has failed or simply been moved out of network range. Devices can be stolen. And a device may be the only one serving an area, so it cannot fail over to another device if there's a problem or it needs to go offline for an update.

To bring container technology to embedded applications, Wind River® has created VxWorks® containers. Instead of using a traditional container, a VxWorks container is a container image that can be used to accommodate the real-time needs of embedded applications. At a high level, a container is a runtime environment plus applications. A container image is a package that contains everything needed to create the application, including libraries and executables (i.e., tools).

VxWorks containers were designed to follow the same OCI specifications used by other container technologies (see Figure 3). This allows them to be managed by the same standard tools that cloud-native developers are already familiar with, such as Docker for adding a container to a registry. However, the container image also includes parameters for how to run the real-time process (RTP), including RTP stack size, priority, and other real-time considerations.

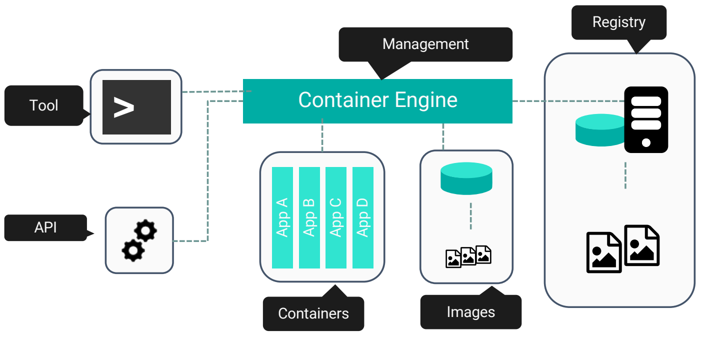

Figure 3. Container delivery

Shell commands are supported to allow developers to access containers from the VxWorks command prompt. While this could be used in a production environment, Wind River has built in a C-based API. This powerful API allows embedded development and deployment teams to manage systems with their existing tools. It also supports services such as programmatic pull, create, kill, etc., giving teams tremendous flexibility in how systems are deployed and managed.

A mount namespace is supported as well, enabling applications and containers to be kept separate from each other and managed independently. This is useful when applications employ different versions of software within the same container. In addition, as devices at the edge become more compute capable, there is the opportunity to consolidate different workloads onto the same device. This leads to mixed-criticality systems that perform before critical and noncritical tasks. Containers provide a powerful way to manage the complexity of mixed-criticality systems by isolating the different components from each other.

PERFORMANCE AND CYBERSECURITY

Wind River has kept the requirements of embedded applications top of mind while developing VxWorks containers. The size of the VxWorks container engine is 353 KB, including all dependencies. The runtime itself is under 100 KB, making it suitable for real-time systems. The OCI container images supported by VxWorks are compressed archives of the application you wish to run inside the container. The runtime will decompress and unpack the image to create a bundle when it is pulled from the container

WNDRVR

registry. While this takes time to complete, it only needs to be done once. Containers can be started quickly from a bundle and can even run multiple copies at the same time. There is also support for overlays so that if multiple containers are using the same library, only one copy need be stored.

VxWorks containers provide hard real-time performance and can run certified real-time processes. By design, the effect of running an RTP in a container introduces very little overhead, introducing negligible impact on performance compared to a native runtime.

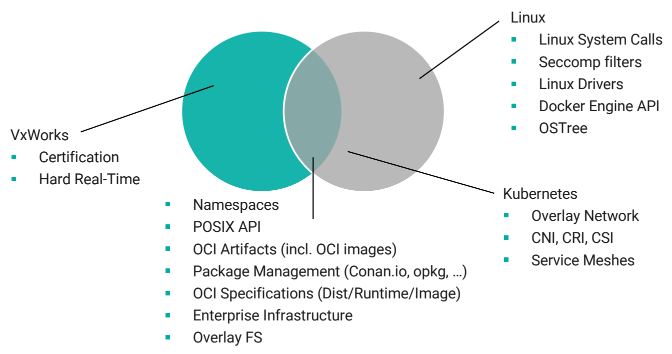

Figure 4. The embedded capabilities of VxWorks containers compared to Linux containers

Figure 4 shows the embedded capabilities of VxWorks containers compared to Linux and Kubernetes containers. Note that many of the Linux containers' capabilities not supported by VxWorks containers are not relevant to embedded applications.

Security is an especially important consideration for connected embedded systems at the edge. To protect systems, VxWorks containers do not expose container management to the network by default. Using the C-based API, developers can ensure that only trusted tools and methods have access. When containers are downloaded from a registry, a system can do this only from trusted registries, with the entire communication encrypted using a secure TLS connection. Furthermore, containers can be signed to ensure that they come from a trusted source.

VxWorks also employs secure boot technology so that a chain of trust can be established. This is achieved by validating all the software components, from the system's hardware root of trust all the way up to the application.

CONCLUSION

Containers make it possible to automate parts of the software development and deployment process. VxWorks containers enable developers to create, test, deploy, and manage embedded systems faster and more accurately. They accelerate development today and will enable the engineers of tomorrow to continue to maintain these systems over their extended lifecycles.

Learn how containers can accelerate your embedded application development: www.windriver.com/products/vxworks/evaluation