LINUX CONTAINERS
Where Enterprise Meets Embedded Operating Environments

WHEN IT MATTERS, IT RUNS ON WIND RIVER
EXECUTIVE SUMMARY

Flexible and connected platforms are core components in leading computing fields, including Network Functions Virtualization (NFV), Software-Defined Networking (SDN), and the Internet of Things (IoT). These modern systems and applications drive requirements on the core platform that lead to the convergence of enterprise and embedded operating environments. As a result, existing methods for build, deployment, and maintenance of embedded platforms are augmented with enterprise-like functionality. Similarly, traditional desktop and enterprise distributions are adopting features previously found in embedded systems.

Traditional embedded devices provide an image-based deployment, application, and service delivery environment. The rise of microservices and the extension of development and operations (DevOps) deeper into the application stack require platforms that can provide an efficient and dynamic environment. This white paper discusses how the platform itself can embrace and extend cloud, container, and microservices for both applications and system services.

Wind River® Pulsar™ Linux bridges the gap between embedded and enterprise platforms to provide familiar containers and virtual machines while maximizing limited resources. This paper examines the container architecture of the platform, its challenges and features, and how it is optimized for the underlying device.

TABLE OF CONTENTS

Executive Summary ................................................................. 2
Embedded Containers ........................................................... 3
  Build Up vs. Tear Down vs. Renovate ................................... 3
  Containers in the Yocto Project Ecosystem ............................. 3
Containers As Building Blocks .................................................. 3
Platform Use Cases for Containers .......................................... 4
  Unleashing Containers in Pulsar .......................................... 4
Platform Services in Pulsar ....................................................... 5
  Coexistence with Virtual Machines (VMs) ............................ 5
Conclusion .............................................................................. 6
As more IoT services are pushed to the edge of the network or onto smaller devices, the need for flexible, connected platforms is on the rise. New open source technologies such as containerization are bridging the gap between embedded and enterprise platforms to meet emerging IoT requirements.

**EMBEDDED CONTAINERS**

Containers operate the same way in an embedded (i.e. resource-constrained) environment as they do in a larger enterprise counterpart. As a result of the facilities provided by the host system, they provide a lightweight virtual environment that groups and isolates a set of processes and resources (such as memory, CPU, disk, etc.) from the host and any other containers.

The technologies that underpin containers are mature and available in almost any modern Linux kernel (e.g. 3.14+). As a result, the use of containers is possible in both existing and new platforms. While the technology itself isn’t new, innovation is coming from its application to the frameworks and platforms.

The question to ask, then, is not whether containers will run on a given platform, but how best to leverage containers for the platform at hand.

**Build Up vs. Tear Down vs. Renovate**

There are usually three options when composing containers for a system:

- Using an existing user space unchanged
- Modifying an existing user space
- Building a new user space from scratch

In addition to selecting the way container user space should be created, questions about how to construct it are equally important (i.e. from source, from binaries, via scripts, via a build system, on target, off target, etc.).

Container support and maintenance issues must be addressed. For example: Can an existing enterprise distribution be streamlined? Should an existing minimal container OS be scaled up? Should binaries or source be used to construct containers? Is the resulting user space supported? Is it certifiable? Are there tools to audit the container userspace? Is it truly minimal and understood?

**Containers in the Yocto Project Ecosystem**

The Yocto Project ecosystem provides all the necessary components to create rich, end user container environments, as well as the ability to create minimal, application-specific environments. Any container created from the Yocto Project can be audited and contains only the applications, libraries, and facilities needed to meet the dependencies of the end user application.

The meta-virtualization layer provides the necessary components and runtime to support Linux containers (LXC), Docker, and Open Container Initiative (OCI) containers. This flexibility means a container runtime that meets project requirements can be used, as opposed to having one preselected.

Containers can leverage a stable ecosystem and use mature technologies to meet the performance and resource constraints of a modern embedded platform. Existing frameworks can be used to manage these containers, resulting in a rich runtime that looks, feels, and behaves like an enterprise-class virtualized environment but respects the unique constraints of the system.

**CONTAINERS AS BUILDING BLOCKS**

The properties of containers mean they can either be building blocks or part of the foundation of many different embedded systems. Containers can meet the requirements of different verticals (IoT, NFV, telecommunications, industrial, enterprise, etc.) as well as run on many architectures (ARM®, IA-32) and classes of device (enterprise systems, gateways, edge devices, embedded/wearable devices, etc.).

Containers provide the implementation of higher-level concepts in a format that is easier to manipulate than raw packages.

Here are samples of building block types containers can provide:

- **Security**: Separation, isolation, resource control, syscall limitation, etc.
- **Packaging**: Library separation, host independence, standardized user space, no package formats (rpm, deb, etc.)
- **Deployment**: Container push/pull via REST APIs, agents, or management frameworks
- **Lifecycle**: Container “atomic” upgrades and rollback
- **Development**: Automated container creation, push, test cycle
- **Portability**: Self-contained user space with only dependency on the kernel application binary interface (ABI)

Having established the benefits of containers, the next section examines where they can be used.
PLATEFORM USE CASES FOR CONTAINERS

Due to their lack of overhead and hardware dependency, containers can be leveraged by nearly any platform. Applications and services provided by the system dictate how containers are used. Taking the end user point of view, we can derive requirements to determine where containers make sense.

Since containers run on every architecture, a simple rebuild and deployment may be all that is required for migration. Nonetheless, there could be additional hardware requirements containers need to support.

Some examples:

- Huge translation lookaside buffer (TLB)–backed containers
- Protected direct memory access (DMA) to containers
- High-performance I/O
- Trusted boot or key management

Common requirements not dependent on hardware:

- Real-time performance
- Application density (i.e. footprint reduction)
- Elasticity (fast boot, fast startup)

Since they can be extensively tuned and tailored, embedded systems have traditionally been an excellent way to meet these sorts of requirements. The tradeoff is that consistency between targets, ease of use, deployment, and management may not be at the level of a binary/enterprise operating environment.

Wind River has bypassed the constraints and unleashed the power of containers in Wind River Pulsar Linux: a small, high-performance, secure, and manageable Linux distribution designed to simplify and speed embedded and IoT development projects. By combining containers with the robust code of Pulsar, Wind River delivers a solution that bridges the gap between embedded and enterprise computing. The platform can be extensively tuned and optimized while reaping the benefits of a consistent base.

Unleashing Containers in Pulsar

As previously mentioned, BitBake, OpenEmbedded, and the broader Yocto Project ecosystem provide a rich build environment, with recipes, packages, and tools to construct platforms, containers, and applications.

Pulsar leverages this foundation to create a partitioned system based on separating the maintenance and operation of the platform from the execution of end user workloads.

Figure 1. Pulsar architectural diagram

Pulsar’s container-based runtime is collectively referred to as “OverC.” OverC provides building blocks (i.e. cubes) that implement a set of related functionality. The current cubes are:

- **cube-dom0**: Provides control services to the platform and is responsible for launching other containers in the system. Dom0 is not manipulated by non-admin users.
- **cube-desktop**: Provides full enterprise-like functionality, including main pages, development tools, graphical desktop, and so on. This domain is the primary environment for most users.
- **cube-server**: Provides a full-featured, headless server environment, similar to cube-desktop, but without graphics.

Depending on the selected cubes, the user/application developer runtime varies. OverC also has two essential image types that provide a foundation for the container runtimes. These are:

- **cube-essential**: This minimal runtime is used for both the installer and cube-essential in the fully assembled system.
- **cube-desktop or cube-server**: These reference containers provide an enterprise-like, fully featured runtime suitable for development or application deployment.
It is also important to point out that while the system is partitioned, this partitioning is not visible to an application or service delivery container. The goal is to have containers be pervasive but transparent to the majority of users and services in the system. Only at the administrative level or within the internal management framework are partitions visible.

The critical element of this system is that the base layer (cube-essential) and control domain (cube-dom0) are consistent across devices, while end user containers and system functionality change without the need to reflash or rebuild the base system. This partitioning also means elements of the system can be independently and safely updated, added, or removed.

PLATFORm SERVICES IN PULSAR

Pulsar uses LXC and OCI containers to provide system services, separation, and domains. But the actual container format and manipulations are transparent to the end user. This layer of abstraction allows Pulsar to evolve as the technology around container management is advanced in the ecosystem.

Coexistence with Virtual Machines (VMs)

The ability to run containers of any format is a key feature of Pulsar, which means that Docker is supported from within end user containers. This also means that the co-existence of VMs and containers is a fundamental aspect of the system (i.e. containers can launch VMs, containers can run within VMs, etc.)

A pillar of the platform is allowing the most appropriate solution to be used, as opposed to enforcing a technical choice or methodology.

Pulsar uses containers to deliver platform services and overall system functionality:

- **Separation and isolation:** namespaces, cgroups, and device management ensure containers are isolated from the host and between containers.
- **Security:** Features include partitioning, privilege control, selinux, read only zones, and trusted boot readiness.
- **Update and rollback:** cube-dom0 agents allow the update of the system on a per-container, package, or image basis. The ability to perform a factory reset and rollback changes is provided as a safety feature.
- **Resource control and monitoring:** The essential system and control domains can enforce resource limits on containers and VMs. They also provide a central location to monitor overall system performance and resource consumption.

In addition to standard system services, Pulsar provides building blocks for extended and enhanced services via tools, scripts, agents (with REST APIs), and the ability to cross build or natively build applications and the platform itself. Containers communicate via the network (both internal and external) or through shared memory pipes to the control domain. These communications channels can be used to extend system functionality or chain containers together for advanced service delivery.

Pulsar can also allow a privileged container to directly manage hardware or other aspects of the platform. This ability allows the core Pulsar platform services to be leveraged while allowing the system to be extended without redevelopment or the redesign of existing user space applications and services.

An example of this extension is the cube-gateway/meta-gateway layer on Wind River Open Source Labs. This layer extends the base platform in such a way as to add additional middleware, which is useful on an Internet gateway platform. Management and configuration of the hardware (i.e. Wi-Fi, Ethernet) and software (i.e. configuration interface, packet processing, etc.) are ceded to the gateway container, but the base system is otherwise unchanged.

Figure 2. Containers: A good balance of simplicity and isolation
CONCLUSION

As more IoT services are pushed to the edge of the network or onto smaller devices, the need for flexible, connected platforms is on the rise. Exciting open source technologies are filling the gap between embedded and enterprise platforms to service IoT needs, and containerization is an up-and-coming area of innovation for manufacturers and service providers.

The ability to run containers of any format is a key feature of Pulsar. Built-in container support enables application middleware abstraction by running multiple isolated systems on the same platform. Taking full advantage of this architectural blueprint, you can start your application development in minutes. Supported hardware platforms and news are located at www.pulsar.windriver.com.