5G will be characterized by a much higher flexibility and programmability for all non-radio network segments including SDN, NFV and IoT. Container technology especially Docker™ with MV CGX Linux offers opportunities to next generation network developers.

Virtualization Leveraged:

Building near Zero Latency Ultra Dense (5G) "Network" for "Things" with MontaVista CGX® and Docker™

MONTAVISTA SOFTWARE, LLC.

AUTHORS
Anantvijay Kulkarni, Senior Technical Solutions Engineer
Iisko Lappalainen, Technical Solutions Manager
Divya Vyas, Technical Solutions Engineer
Sachin Kaushik, Technical Marketing Specialist

Contents

Executive Summary

MontaVista® Linux® Carrier Grade eXpress (CGX), delivers Carrier Grade reliability, security, and serviceability that is highly configurable, flexible, and of consistent high quality. CGX meets the demands of the interconnected intelligent devices, providing application portability, dynamic configuration, field maintenance, and real-time performance in a single platform.

CGX addresses a very large embedded device segment including networking and communications, instrumentation and control, aerospace and defense, SOHO devices, medical electronics and the "Internet of Things (IoT)" markets.

This paper describes how Docker interoperates with CGX. Docker is the leading container-based technology that offers a more efficient approach to application deployment. Together with CGX, it creates a powerful solution for key use-cases in the networking realm. Many of these use cases will be covered in more detail later in this whitepaper.

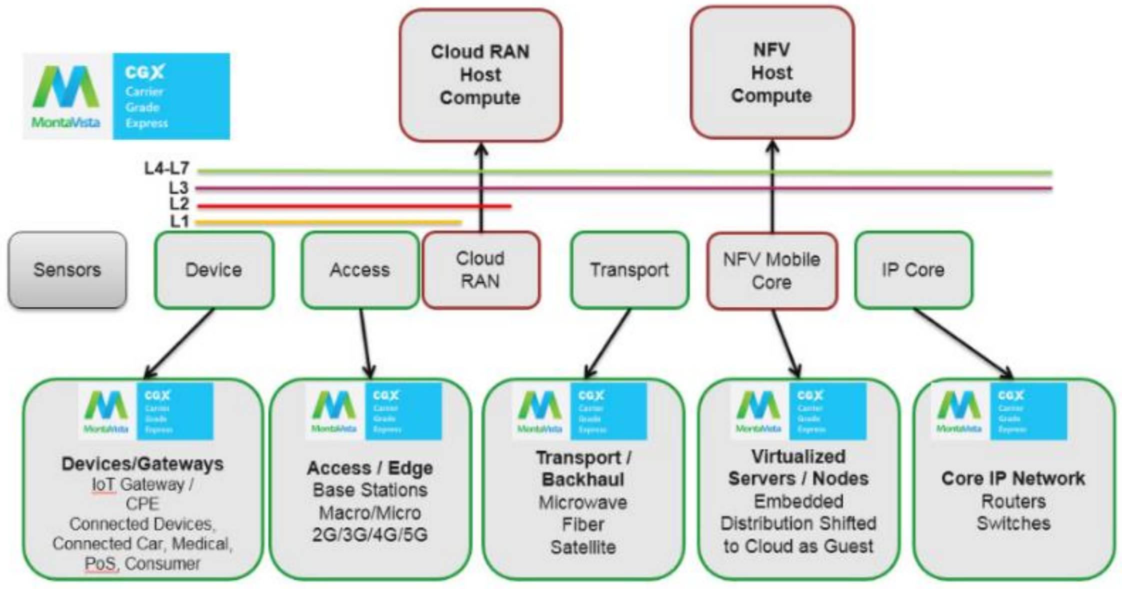

Fig 1: MontaVista CGX Solution Coverage Applicability

Introduction

Network Equipment Providers and Telecommunications carriers are under pressure to deliver higher performance networking at lower cost of ownership. They need to increase operating efficiencies while lowering power consumption, in effect becoming more 'green'. At the same time, they must achieve near 100% uptime as they re-architect their solutions away from proprietary hardware and software to new multi-core architectures to achieve the necessary performance and cost savings.

MontaVista Carrier Grade eXpress (CGX) supports multiple options for maximizing the resource utilization of multi-core processors, with both AMP and SMP support, along with new partitioning and virtualization technologies. MontaVista provides three methods of virtualization to offer the best combination of flexibility, performance, and ease of application development. All are based on non-proprietary, open-source Linux technology and are supported by MontaVista across multiple processor architectures. The three methods are:

- KVM Hypervisor—Full virtualization with para-virtualization options.
- Linux Containers—Operating system resource virtualization.
- Core Isolation—Hardware partitioning.

This paper describes how Docker interoperates with CGX. Docker is the leading container-based technology that offers a more efficient approach to application deployment. Together with CGX, it creates a powerful solution for key use-cases in the networking realm. Many of these use cases will be covered in more detail later in this whitepaper.

The following types of applications are well-suited for the Embedded Docker technology:

- Applications that have low-latency requirements.
- Applications with performance requirements that depend on low level technology of the hardware platform.

Other types of applications that are suitable for CGX with Docker are:

- Applications that requires a plug-in type or dynamic provisioning.
- Applications that are part of a larger pool of applications that are all being consolidated on a single platform. Although full virtualization is commonly used to solve this type of problem, CGX with Docker is a much more efficient approach.

Solution Overview

Docker™ and MontaVista CGX

MontaVista has been commercializing Linux Containers (LXC), an implementation of container based virtualization technology, for a long time. Linux Containers have been a part of the product since the introduction of the fifth (5th) generation MontaVista Carrier Grade Linux distribution since 2009. Docker™ is built on the same containers technology.

CGX provides a strong foundation for Docker. Combining Docker with CGX offers several advantages over plain desktop distributions:

- CGX is 100% native Linux with real-time performance features, including integrated high resolution nanosecond timers (hrtimers) and other MontaVista enhancements.

- CGX is a true multi-architecture platform with the latest 4.x kernel and toolchain. Also MontaVista is the only vendor to provide true support for ARM64. This underlying multi-architecture support enables Docker across all these same architectures.

- MontaVista has been commercializing various virtualization technologies, such as KVM and LXC, for some time. With the addition of Docker to CGX, you can choose the right virtualization technology for the right problem.

- With the isolation techniques in CGX, Docker provides a powerful way to distribute applications to high-performance domains. These domains isolate the application from the rest of the Linux operating system, providing performance unhindered by the operating system. Both applications with low real-time latencies and applications with high packet performance characteristics can benefit from these isolation techniques.

- MontaVista provides commercial support for CGX and offers services to customize CGX for your specific requirements. MontaVista also provides support even when there are modifications to the baseline, by either the customer or MontaVista Professional Services.

Docker, with CGX, defines an image format, provides an API for container management, and supports remote registries for sharing containers. The benefits of this scheme are:

- **Portability across machines—**Docker lets you create a single object containing all your bundled applications. This object can then be migrated to any other Docker-enabled host. With Docker, you get:
    - A containers-based virtualization solution suitable for dynamic multi-node cloud deployments.
    - Live Migration capabilities.

- **Security and Isolation of services and applications—**Each container runs in its own sandbox and is completely isolated from other containers so that you can
    - Comply with legal or contractual obligations to isolate an application.
    - Prevent flawed applications from compromising the rest of the system.

- **Limit resource usage—**Docker on CGX makes it easy to limit resources, such as CPU and memory, available to an application. As a result, you get higher density and run more workloads.

- **Application-centric, easy and fast removal and addition—**Docker on CGX is optimized for the deployment of applications rather than machines. You can create fifty containers in just seconds.

- **Copy-on-write mechanism—**Your filesystem is shared with other containers. When something needs to be written out, the container makes a copy of the file and stores it in a separate location, resulting in:
    - Better utilization of system memory.
    - Higher density of containers for a given resource than other container implementations.

- **Version control—**Docker includes git-like capabilities to track successive versions of a container. Docker lets you inspect the diffs between versions, commit new versions, and roll back to previous versions.

- **Container Repository—**it is possible to build a library of readily available and re-usable base images.

- **Component reuse—**Docker lets you build or stack already created packages. For example, if you need to create several machines that all require Apache and MySQL, you can create a base image that contains these two items, then build and create new machines using these already installed. The benefits are:
    - Reducing the cycle time of development, testing and deployment
    - Easy to deploy PaaS-type solutions (pls. refer to use-cases)

- **Active Community—**MontaVista CGX has a substantial user base, with more customers joining rapidly. The Docker community is active and growing very fast. The combination of these two communities is resulting in a wave of development that will provide even more value around the CGX and Docker technologies.

Linux Containers Technology and Docker™ Technical Background

The Linux kernel comprises of cgroups for resource isolation (CPU, memory, block I/O, network, etc.) and namespace isolation to completely isolate an application's view of its operating environment.

Containers combine cgroups and namespace support to provide a virtual environment that has its own process and network space, instead of creating a full-fledged virtual machine (see Figure 2).

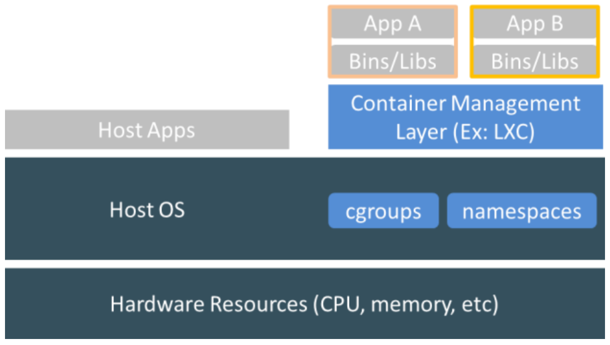
App A
Bins/Libs
App B
Bins/Libs
Host Apps
Container Management Layer (Ex: LXC)
Host OS
cgroups
namespaces
Hardware Resources (CPU, memory, etc)

Figure 2: Linux Containers

Docker is an open source engine that enables any application and its dependencies to be packaged as a lightweight, portable, self-sufficient container that can run on any Linux machine. This removes any restrictions on where the application can run. It can be easily run on premise, public cloud, private cloud, bare metal, and so on. Docker represents a different model for virtualization (see Figure 3) than the traditional hypervisor virtual machine (VM) approach espoused by VMware's ESX, Microsoft's Hyper-V, Xen and the open-source KVM (Kernel-based Virtual Machine) technologies.

With a traditional hypervisor, each VM needs its own operating system. In contrast, with Docker, applications sit inside a container that resides on a single host operating system that can serve many containers.

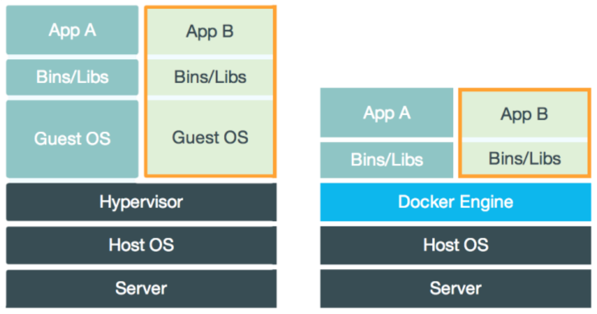

Figure 3: Hypervisor v/s Docker

Containers are a complex technology and require a complex set of configuration and command line use. Docker provides a simple interface to manage your containers. Docker with its V0.9 release introduced an execution driver API which can customize the execution environment surrounding each container. This lets Docker take advantage of the numerous isolation tools available, including: OpenVZ, systemd-nspawn, libvirt-lxc, libvirt-sandbox, qemu/kvm, BSD Jails, Solaris Zones, and even chroot (see Figure 4). This is in addition to LXC, which continue to be a supported driver.

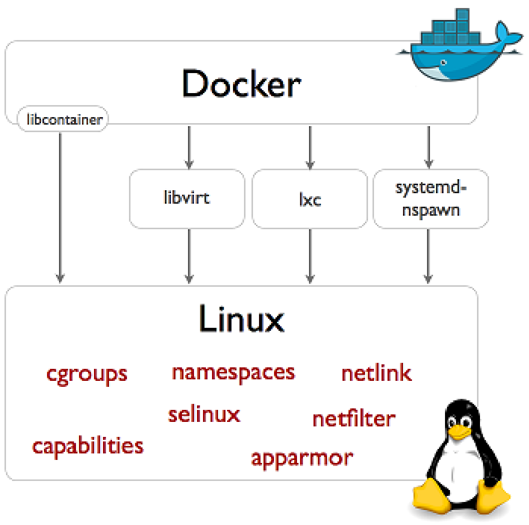

Figure 4: Technologies underneath

Docker introduced a new built-in execution driver based on libcontainer, a pure Go library (see Figure 4). Docker can now manipulate namespaces, control groups, capabilities, apparmor profiles, network interfaces and firewalling rules, all in a consistent and predictable way, and without depending on any other userland package.

This drastically reduces the number of moving parts, and insulates Docker from the side-effects introduced by LXC. In fact, libcontainer delivered such a boost to stability that it was made the default execution driver. Thus Docker is a collection of low-level technologies that already existed but organized in a way that is more than the sum of its parts.

Docker has two major components:

- **Docker Engine**—Core software platform that enables users to create and use containers.
- **Docker Hub**—SaaS-based service for creating and sharing Docker services. With the release of the 1.0 version and Docker Hub, the company has more than 14,000 applications that can be used with its containers.

How secure are Docker containers?

There are three major areas to consider when reviewing Docker security:

1. Intrinsic security of containers, as implemented by kernel namespaces and cgroups
2. Attack surface of the Docker daemon itself
3. "Hardening" security features of the kernel

Let's look at each of these areas briefly.

1. Intrinsic security of containers

Docker uses two primary technologies in Linux to make containers possible; these include kernel namespaces and cgroups.

Kernel Namespaces

Namespace isolates an application's view of its operating environment. Processes running within a container neither see nor affect the processes running in another container, or in the host system. Also each container gets its own network stack, meaning that a container doesn't get a privileged access to the sockets or interfaces of another container.

Control Groups

Control Groups ensure that each container gets its fair share of memory, CPU, disk I/O; and, more importantly, that a single container cannot bring the system down by exhausting one of those resources. Thus mitigating the denial-of-service attacks which are of serious concern on multi-tenant platforms, like public and private PaaS. The code for kernel namespaces is available since 2008 and the code for control groups has been around since 2006. Thus the namespace and cgroups code has been exercised and scrutinized on a large number of production systems.

2. Security of Docker Daemon

Running containers (and applications) with Docker implies running the Docker daemon. This daemon currently requires root privileges, and you should therefore be careful. Thus only trusted users should be allowed to control your Docker daemon.

However it is to be noted that Docker is implementing two additional security improvements:

- Map the root user of a container to a non-root user of the Docker host, to mitigate the effects of a container-to-host privilege escalation;
- Allow the Docker daemon to run without root privileges, and delegate operations requiring those privileges to well-audited sub-processes, each with its own (very limited) scope: virtual network setup, filesystem management, etc."

3. "Hardening" security features of the kernel

There are many different ways to harden a Docker host. Let’s look at a few of them.

Linux Kernel Capabilities

Linux Kernel Capabilities turn the binary "root/non-root" dichotomy into a fine-grained access control system. By default, Docker starts containers with a very restricted set of capabilities. And there are capabilities, for almost all the specific areas where root privileges are usually needed. Thus the "root" within a container has much less privileges than the real "root". This means that even if an intruder manages to escalate to root within a container, it will be much harder to do serious damage, or to escalate to the host.

Kernel with grsecurity and PaX

This will add many safety checks, both at compile-time and run-time; it will also defeat many exploits, thanks to techniques like address randomization. It doesn't require Docker-specific configuration, since those security features apply system-wide, independently of containers.

Linux Security Modules

Linux kernel supports various security modules like AppArmor, SELinux, etc. Providing a security policy/template for Docker containers boosts up the security with an extra safety net (even though it overlaps greatly with capabilities).

Security in CGX

Networking and Carrier technology is an area where security breaches have a very critical impact and attackers have much to gain to gaining access to these systems. CGX is a technology baseline with hardened and trusted security measures, policies and practices

Let's look at security from CGX point of view.

Standards Conformance

CGX supports various security standards.

- **CGL** - Carrier Grade Linux specification defines a baseline for Carrier Class Linux Distribution. It also outlines the various security requirements like role based access control, system auditing and log collection, etc.
- **STIG** - Security Technical Implementation Guide is a methodology for standardized secure installation and maintenance of computer software and hardware.
- **USGv6** - It is IPv6 security profile for US Government. NIST developed a technical standards profile for US Government acquisition of IPv6 Hosts and Routers, and a specification for Network Protection Devices.

- **OSPP** - Operating System Protection Profile defines security functionality expected to be provided by an operating system in a networked environment.

"Hardening" security features of the kernel

The security of the CGX kernel is hardened by various means like grsecurity, Linux security modules, Linux capabilities, etc.

PaX is a major component of the grsecurity. PaX provides protection by utilizing a strategy to provide only the minimum privileges to memory pages restricting programs memory access rights to do only what is required to execute properly, and nothing more. This effectively prevents many security exploits, such as buffer overflows.

With **Linux capabilities**, the system is divided into logical groups that may be individually granted to, or removed from, different processes. This providing a fine-grained control over the system and enhances the system security.

Linux Security Modules (LSM) is a framework that allows the Linux kernel to support a variety of computer security models. SELinux is a Linux kernel security module that provides the mechanism for supporting access control security policies. CGX provides a customizable default SELinux policy for its customers.

CVE - Common Vulnerabilities and Exposures

The CVE system provides a reference-method for publicly known information-security vulnerabilities and exposures. MontaVista pays for membership into groups that provide early warnings of CVE's before they go public. MontaVista researches all of the CVE's that might be relevant to our distribution and it becomes a top priority for our engineering team to find or create a patch and get it fixed before our next release date. This is an expensive and time consuming process, so our customers don't have to.

Wide Deployment

MontaVista Carrier Grade Express is based on one of the most widely deployed Carrier Grade Linux. Hundreds of customers and thousands of engineers are constantly pushing the envelope with our Carrier Grade Edition, hitting the corner cases and finding bugs that we fix and pass on to all of our customers. Since we a unified kernel; this means that all major architectures are under one kernel tree and when we fix a bug that we find on one architecture, it is fixed on all of the architectures.

USE CASES

The following sections describe applications that could benefit from CGX and the Docker:

- Platform-as-a-Service Cloud.
- Containers-Based Multi-Tenancy in the Cloud.
- Bundling or Consolidating HW+SW Configurations in Network Servers.
- Migration between Legacy Virtualization and Containers.
- Cloud RAN.

Platform-as-a-Service (PaaS) Cloud

One of the main use cases driving the containers-based virtualization is the PaaS cloud. Compared to traditional hypervisor-based virtualization, Docker can significantly improve the scalability and performance characteristics of running a large number of application stacks on a given set of host servers.

The Platform-as-a-Service cloud is depicted in Figure 5. It also shows an option to use OpenStack to manage the Docker containers hosting both the PaaS supporting stack and the Application. The Docker model where several containers can be stacked up on top of each other is an excellent fit for the PaaS use case.

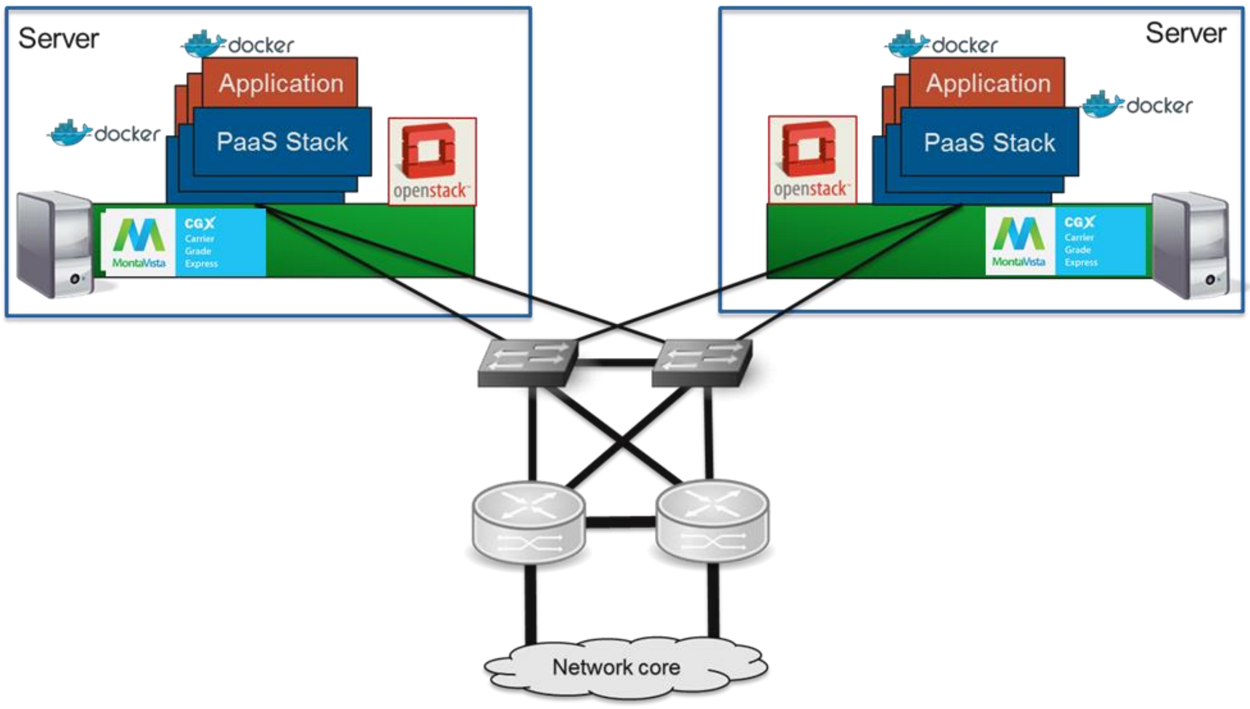

Figure 5: PaaS Cloud with CGX and Docker

Containers-Based Multi-Tenancy in the Cloud

At MontaVista, we see a theme around the domain separation and Cloud virtualization which appears to start from bare-metal –type virtualization, then evolve to a hypervisor-based virtualization with the Cloud model, and then re-emerge as a version of the initial model as performance requirements force optimizations to the hypervisor solution.

The original usage models from the bare metal domain separation used in earlier control systems are implemented in the hypervisor-case as an abstraction layer that presents a common API to access the HW resources that are controlled by the hypervisor.

However, in several cases the hypervisor-provided approach is not completely suitable and pass-through options are used to directly control HW resources and peripherals for performance reasons (network device bypass via VFIO, SR-IOV, Direct memory access etc.). In many of these cases the bare metal performance requirements are better served by using a model that has more direct access to the operating system resources – as Docker in CGX has.

What make this re-emergence now possible are the characteristics of Docker virtualization that allow migration of the applications dynamically in the Cloud setup, like with virtual machines. It is also possible to customize a solution that provides Live Migration capabilities with CGX and Docker, making it analogous to VMs also in this regard - both are key capabilities not offered by the initial domain separation models.

Figure 6 below depicts a multi-tenant router solution built on CGX and Docker, with using OpenStack to control the initialization of the tenant domains.

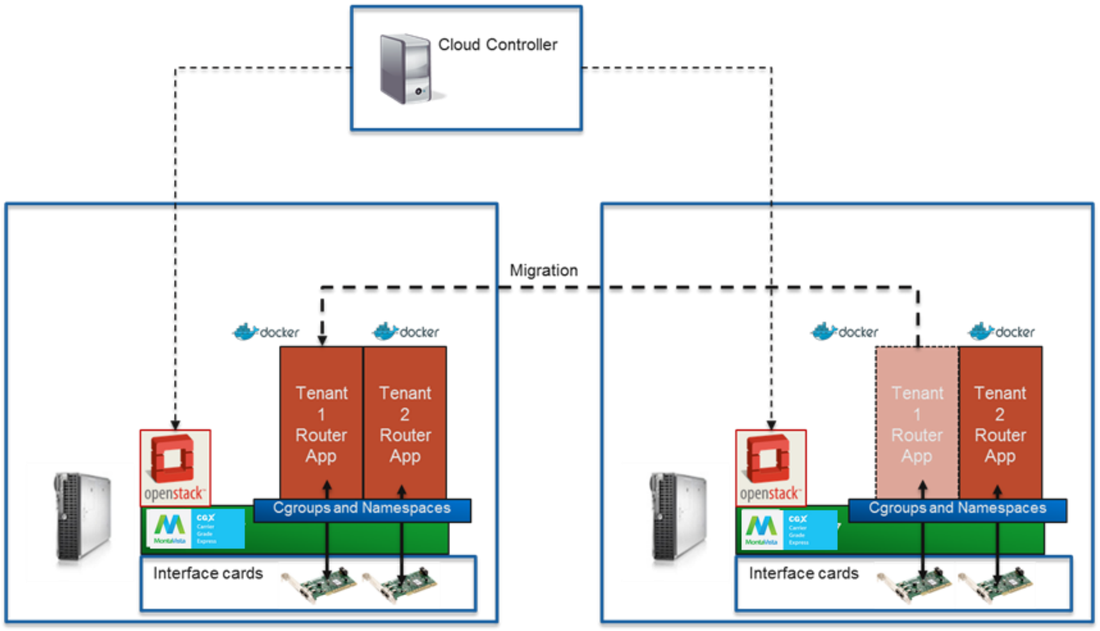

Figure 6: Multi-tenancy with Containers

Bundling or Consolidating HW+SW Configurations in Network Servers

Several server build models have hot pluggable HW components that in some cases require SW support to be served correctly on the hosting OS. Such as ATCA Blades + AMC cards, Standard Servers with Offload cards etc. The same use case can also be utilized when there is a requirement to consolidate certain legacy applications all on the same platform. In this case the containers can provide domain separation for the different application sharing the platform, like using a hypervisor-based approach.

Using Docker containers we can envision a model that allows hot-plugging the device itself, then automatically retrieving the Docker container associated with the device from a known pool of container images containing the necessary supporting software.

Figure 7 illustrates this use case.

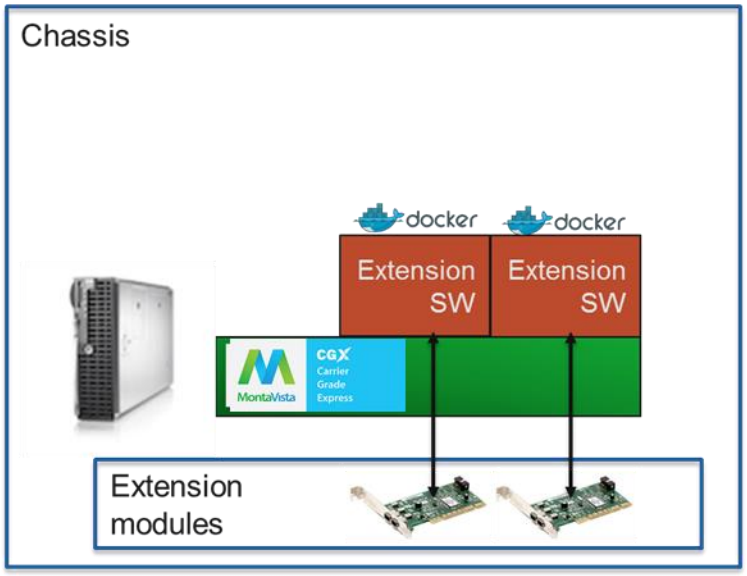

Figure 7: Using Docker to support HW+SW plugins

Migration between Legacy Virtualization and Containers

At MontaVista, we believe that one of the unique features CGX can do with Docker to provide significant value is the ability to move applications to and from KVM Hypervisor-based applications to Docker-based application contained in either virtual machines or containers domains.

In principle the user-case is based on Docker compatible kernel support in the VMs, and the ability to encapsulate the application part into Docker which can be migrated to a platform with specific suitability for embedded virtualization, even HW platforms that don't support hypervisor-based virtualization well - like ARM platforms in the Mobile Radio Infrastructure.

Figure 8 depicts this use case where an application running virtualized on a COTS-based server can be migrated to an embedded architecture where it can run on bare metal. To be noted, the embedded kernel components typically don't run well on top of a hypervisor so full OS on the embedded target would not be runnable as a full VM.

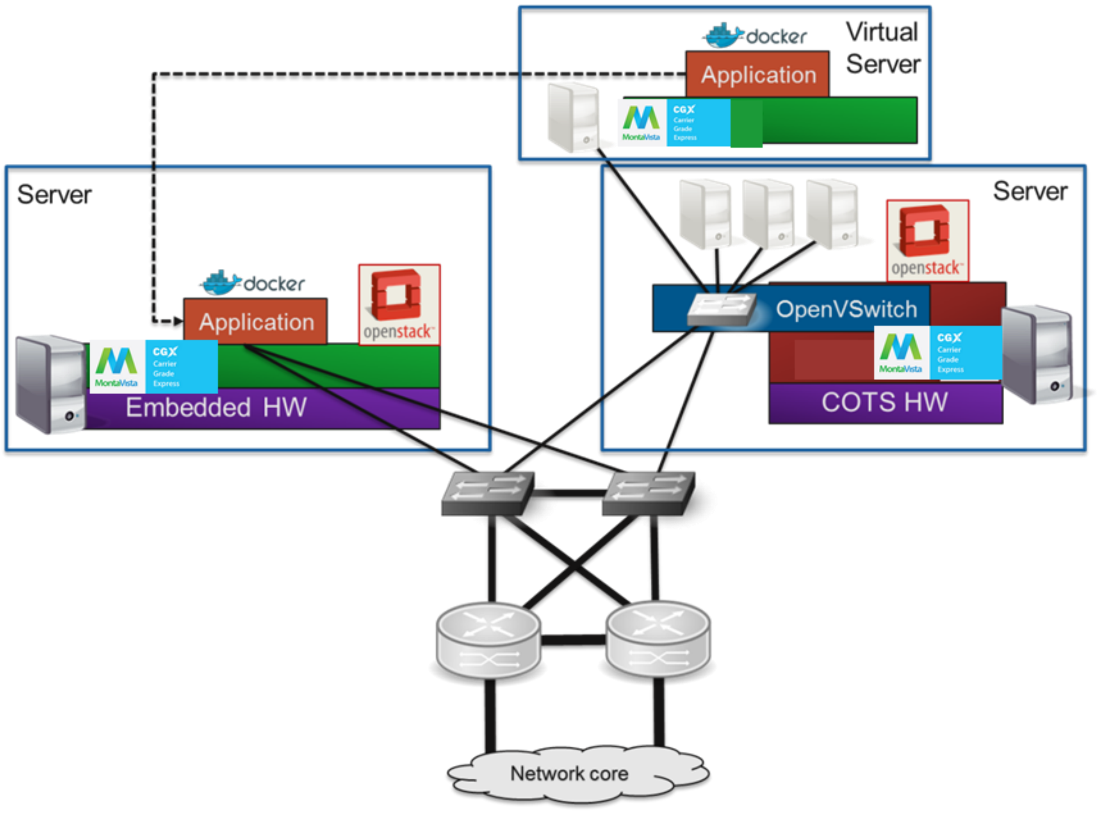

Figure 8: Application-level Migration with Docker

Cloud RAN

The term Cloud Radio Access Network (Cloud RAN or C-RAN, also Centralized RAN) refers to a specific deployment in NFV where the standard base stations of a telecommunications network are separated into a virtualized baseband processing units (BBU) and the antenna systems, known as Remote Radio Heads (RRH). The BBU in pools provide consolidation for the processing of the baseband functionality, while the RRH themselves contain the first stage of processing, converting the analog radio signal to digital.

In essence this creates a layered architecture where the first tier of processing for the RF layer is handed in the RRH, but the MAC and upper layers are handled in the BBU pool.

This is a particularly challenging scenario, since the baseband processing has strict real-time requirements and requires very high bandwidths. Also while the target is to use commonplace HW in the BBU pools, there usually is a need for some customization. This is because the platforms typically contain custom and non-mainstream HW at least to some extent, for example for handling the links to the RRHs.

In Figure 9 we see a use case using CGX as the hosting OS and using Docker containers as the domains to host the BBU SW. This provides a significant advantage over hypervisor-based virtualization in that the Docker containers don't suffer any significant performance penalties compared to bare-metal applications.

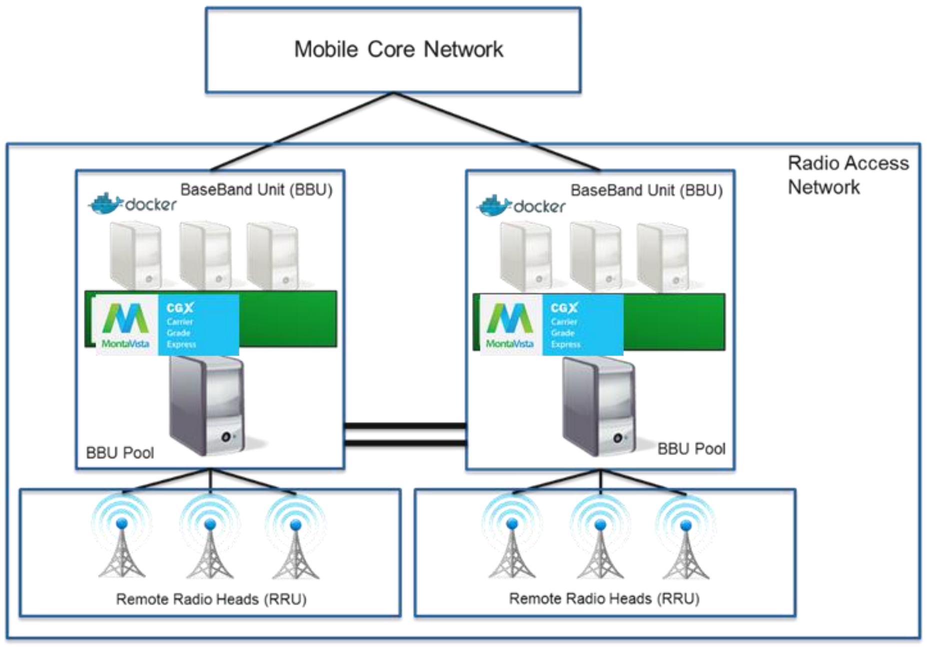

Figure 9: Cloud RAN with CGX and Docker

.

Conclusions

MontaVista Software has been one of the founding member and key contributor to Carrier Grade Linux. MontaVista Carrier Grade Edition distribution has been in the market for over 14 years and the latest generation Carrier Grade eXpress (CGX) leverages this expertise and a number of technologies for maximizing resource utilization of multi-core processors. CGX delivers some compelling advantages:

- 100% native Linux with real-time performance.
- True multi-architecture platform with support for ARM64.
- Long term commercial support cycle with customizable models for different use-cases.
- Support for various virtualization technologies.

With the support of Docker in CGX, you can now benefit from the powerful combination of Carrier Grade Linux and container-based virtualization to build more efficient systems. Many of the use cases discussed in this whitepaper have thus far been driven by hypervisor-based virtualization. With CGX container virtualization, you can benefit from significant enhancements in scalability, maintainability and performance. The use cases best handled with container-based virtualization are:

- PaaS Cloud
- Cloud RAN
- Migration between Legacy Virtualization and Containers
- Bundling or Consolidating HW+SW configurations in network servers
- Containers-based Multi-Tenancy in the Cloud

MontaVista has been a leader in powering Millions of embedded device successfully since 1999. We continue to provide even more advanced solutions in the embedded Linux and virtualization domain, with the inclusion of Docker.

Appendices

Appendix A: Benchmarks

The following list shows the Docker performance characteristics.

1) **Asynchronous launch**
   Test case: Launch 15 Docker Containers / KVM machines continuously without break
   - Time to launch: 4x faster
   - Max CPU utilization during launch: 3.5x lesser CPU utilization
     - Other processes running on the same node are unaffected as enough processing power still available
   - Average CPU utilization at the steady state: 9.5x lesser CPU utilization
   - Memory utilization: 6x lesser memory usage
     - Higher Density / more number of Docker containers compared to VMs for a given memory size

2) **Serial launch**
   Test case: Launch 15 Docker Containers / KVM machines one by one, waiting for each container to start before launching another.
   - Average boot time: 1.6x faster
   - Max CPU utilization during launch: 16x lesser CPU utilization
     - Other processes running on the same node are unaffected as enough processing power still available.
     - In case of VMs, the CPU utilization increases linearly with the increase in the number of VMs launched
   - Memory utilization: 4x lesser memory usage
     - Higher Density / More number of containers compared to VMs at a given memory size

3) **Serial reboot**
   Test case: Reboot 15 Docker Containers / KVM machines one by one, waiting for each container to start before rebooting another.
   - Average reboot time: 50x faster
   - CPU utilization during reboot: 2x lesser CPU utilization

4) **Snapshot Container/VM to Image**

Test case: While the container/VM is active, snapshot it to image.

- Time to snapshot: 1.3x faster
- CPU utilization during snapshot: 2x lesser CPU utilization
- Memory utilization: 2x lesser memory utilization

5) **Guest performance**

- CPU:
    - Linux sysbench CPU test
        - Performance of Docker container = Bare metal = KVM machine
- Memory:
    - MEMCPY
        - Performance of Docker container = Bare metal (11% faster than KVM)
    - DUMB
        - Performance of Docker container = Bare metal (27% faster than KVM)
    - MCBLOCK
        - Performance of Docker container = Bare metal (78% faster than KVM)
- Network:
    - Netperf
        - Performance of Docker container = Bare metal = KVM machine

MontaVista Software, LLC | 2315 North 1st Street San Jose, CA, 95131 | www.mvista.com

DOC. ID. (MVWP-Docker-120916)