BUILDING DOCKER CLOUD SERVICES WITH VIRTUOZZO

Improving security and performance of application containers services in the cloud

Executive Summary

Application containers, and Docker in particular, are a revolutionary technology that has a chance to dramatically change how developers will build, deploy, and manage their applications, thus generating demand for new cloud services and creating new opportunities for service providers.

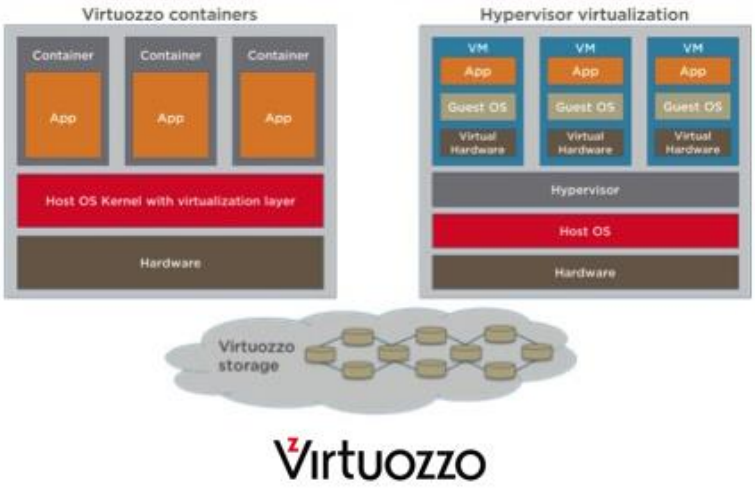
However, being a new and rapidly developing technology, Docker is still lacking the secure multi-tenancy and resource isolation functionality needed for public clouds. This makes Docker-based services expensive to deliver for both the service provider and the end user due to either the higher infrastructure costs required to introduce an extra layer of resource isolation (like running Docker in virtual machines), the need to designate individual infrastructure resources for each customer to compensate for the lack of proper resource management, or the need to invest in building a new management solution around Docker.

Docker integration with Virtuozzo solves most of these problems, allowing service providers to offer Docker-based services on a proven platform that supports the security demands of public clouds with a console for the delivery and lifecycle management of those services. In comparison to virtual machines, Virtuozzo containers provide the same native performance but with higher density that supports more Docker applications per host.

About Virtuozzo and Docker

Docker is an open platform that gives developers a way to build cloud applications, test them against multiple developer frameworks, and run them in multiple environments and on any number of end points. Unlike traditional application management solutions, Docker places an application in a container using the namespace and resource management capabilities of an operating system kernel (Linux) and delivering the application with a full set of OS components (libraries, services) that the application is required to run. This process greatly simplifies the management and deployment of those applications.

Virtuozzo is a high-security, high-performance virtualization platform that combines containers, virtual machines, and software-defined storage to give service providers a single, cost-effective platform for delivering resource-isolated OS partitions and high-availability cloud services to their customers.



Different from Docker application containers, Virtuozzo containers are operating system containers. They allow service providers to securely host dozens – if not hundreds – of customers on a single physical server, often with root access for every customer and often directly connected to the internet, which makes them targets of security attacks simply due to the vulnerable nature of public VPS services. Virtuozzo containers allow their owners to run Docker containers inside them, combining the benefits of "Docker-ized" applications with Virtuozzo security and isolation, but without introducing the extra overhead associated with hypervisors that can negatively affect performance.

Security and Multi-tenancy of Application Containers

A critical capability of any public cloud service is secure multi-tenancy. Commonly, multi-tenancy is achievable either by building multi-tenancy into the service, or by placing multiple instances of single-tenant services into different environments.

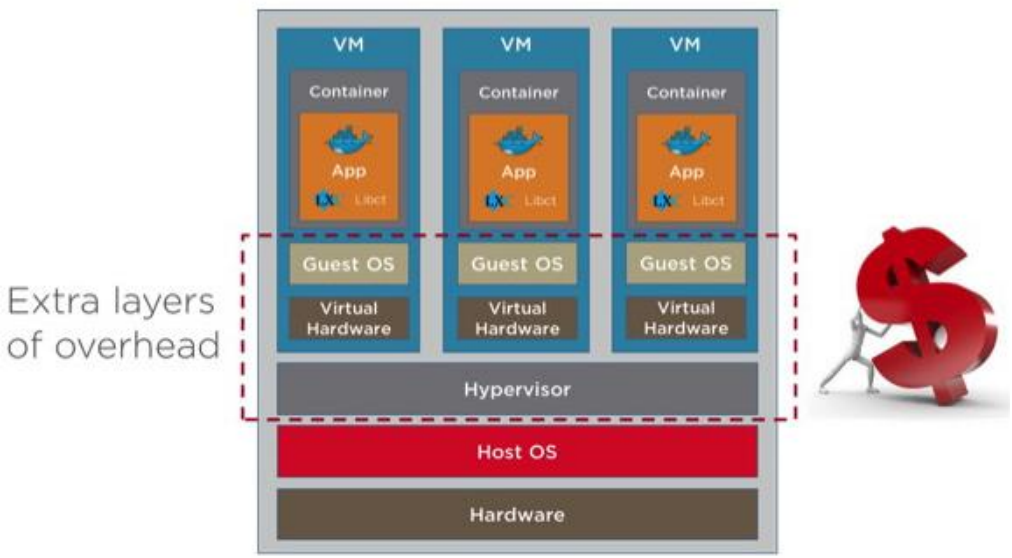
Natively, a Docker-enabled Linux server is not truly multi-tenant. While multiple Docker users can theoretically coexist on a single server, this is not practical in any public cloud where the provider must assume that tenants could be hostile or compromised. An example is a Docker user's ability to escalate his own privileges to root, which increases the vulnerability of all other users on the same server. Like with many new technologies, security vulnerabilities with Docker are discovered quite frequently, and particularly those that allow a process to "escape" from the container gaining real root. (See security advisories like CVE-2015-3627, CVE-2015-3629, CVE-2015-3630, and CVE-2015-3631.)

Lack of native Docker multi-tenancy may be partially remedied by management tools on top of Docker that implement multi-tenancy in the management layer and restrict the tenant's access to the hosts by Docker commands only. But this solution is only partial.

Firstly, implementation of such management tools and their suitability for public cloud – including security, performance, and stability of the API— is yet to be proven. Secondly, from a security standpoint, an architecture that relies on permission from the management layer (that implements tenants) while allowing the tasks to run in privileged mode on the service layer (Docker-enabled Linux host) is considered by many as unsecure by design, essentially amounting to a vulnerability in the management layer that compromises the whole stack.

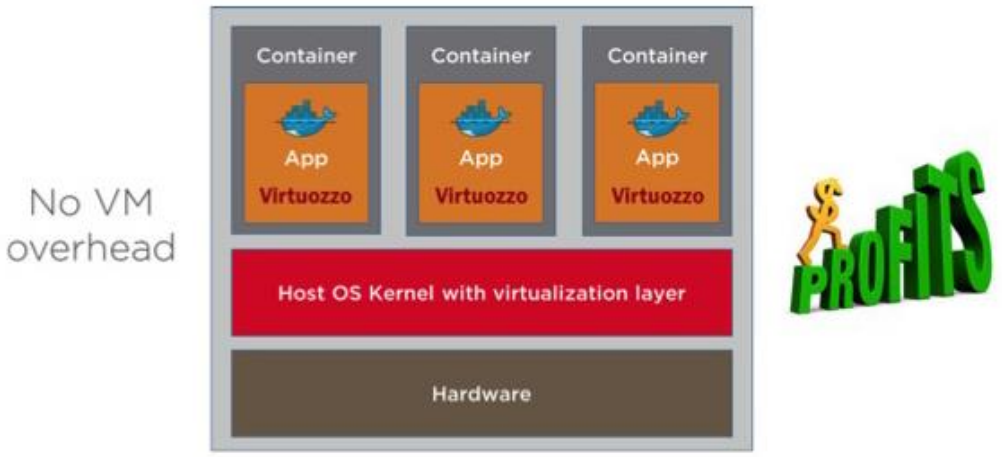
When Docker first entered the market in March 2013, Docker applications worked only with open source LXC containers, which then and even today have insufficient stability, resource management functionality, and security features for public cloud purposes. A year later, Docker dropped LXC as the default container environment and replaced it with containers built using its own libcontainer userspace library. But like LXC, libcontainer has its own security shortcomings since it also wasn't built with the intention of being secure enough for public cloud computing and is still in its early development stages.

These shortcomings can be solved by using a configuration where the open source container holding the Docker application is placed inside a virtual machine, which provides the isolation and security required for running Docker applications safely in a public cloud. By doing this, the virtual machine security characteristics transfer to the Docker applications, so the isolation problem is fixed. However, such isolation is not compatible with the promise of native and bare-metal application performance that Docker can bring because the management efficiency and density advantages that are inherent to operating system container virtualization are eliminated.

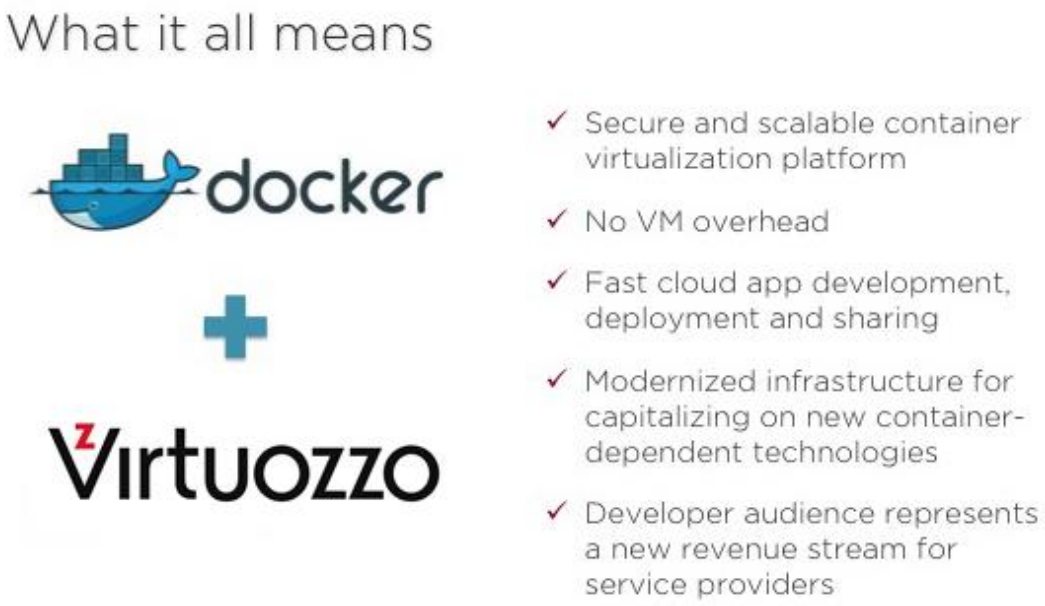


Today, running Docker containers inside Virtuozzo containers fixes most, if not all, of these issues. If a Docker-specific vulnerability is exploited inside a non-Virtuozzo environment, it could possibly elevate the Docker container privileges to the root privileges. However, if the vulnerability happens inside a Virtuozzo container, it would have no impact to the physical host where the containers are running. Docker's lack of multi-tenancy in this scenario is no

longer an issue, because Virtuozzo will isolate the tenants in their own virtual servers, effectively adding the tenancy



There is an additional benefit in this nested containers approach. As many public service providers have already invested in acquiring or building their own management tools for Virtuozzo (and cloud servers, such as VPS), those same management tools can be used to deliver Docker-enabled servers with little or no extra investments.



Resource Management

Quality of services (QoS) is one of the primary outputs that a cloud service must deliver. There are three main goals of resource management in public clouds:

- Delivering guaranteed amount of resources regardless of overall resources usage. Even in a situation of resources shortage, the tenants must receive minimum resources according to their SLAs.
- Managing resource fair shares. When multiple tenants compete for resources above their minimum guarantees, the resource management system must ensure the resources are fairly shared between tenants.
- Isolating a "noisy neighbor." In a public cloud, it is not uncommon to find a tenant that uses too many resources, often in violation of the service agreement. For example, in a platform intended for web hosting, some users might be running CPU-intensive applications that seriously impact the performance of other well-behaving tenants.

Considering these resource management aspects, Docker with Virtuozzo can deliver much better QoS than Docker alone. Let's look at some examples below.

Making Memory

Natively, both Virtuozzo and Docker support limiting the amount of memory available to a container. However, the ways they implement and enforce the memory management are quite different.

In Docker, one can limit the amount of memory available to a single container by adding the "-m" option to the Docker command. This limit corresponds to the "user" memory, i.e., the memory directly allocated by the processes running in a container.

However, a container can also allocate memory differently – e.g., the "kernel" memory (allocated indirectly by making kernel API calls) would not be accounted for. This opens up an easy opportunity for a denial of service (DoS) attack that could bring the service down for other tenants on the same host. Running Docker inside Virtuozzo containers gives the following advantages for memory management:

1. Complete memory management limit that includes kernel memory and swap, effectively preventing DoS (assuming memory limits are set correctly and correspond to available physical memory)
2. Accurate reporting of available memory to an application (natively, a Docker container reports information about physical memory available on the host, and that can make an application attempt to allocate more memory than what is available)

Managing CPU

Docker container CPU resources are managed by assigning CPU share to each container. This ensures that an individual Docker container gets its fair share allocation and – assuming the entire host is not oversubscribed – guaranteed resources.

For running Docker in Virtuozzo, the same type of management is available, along with several other critical features:

3. Virtuozzo can enforce CPU limit on a tenant, solving the "noisy neighbor" problem.
4. Virtuozzo manages resources on a per-system container basis (or per tenant, assuming a ratio of one tenant = one Virtuozzo container). A single container will get its CPU share regardless of how many Docker containers are running inside. In native Docker, CPU shares are managed on a per-container basis – therefore, a tenant running 10 small containers will get 10 times more CPU than a tenant running one larger container.
5. Virtuozzo is NUMA-aware and always attempts to place container processes on a single NUMA node. This typically translates to better overall performance as individual Docker containers are running more effectively and leveraging local memory.

Managing Disk Space and Input/Output Operations per Second (IOPS)

Docker natively has no capability to manage disk consumption for the entire container fleet of a single tenant. Even managing disk quotas of a single container is somewhat

cumbersome because of how these disk quotas are implemented for different storage backends used with Docker.

Together with Virtuozzo, the storage resources problem is much easier to solve:

6. Virtuozzo has disk quotas that are enforced for the entire system container, regardless of how many Docker containers are running inside or what kind of storage backend they use. This makes disk space consumption much easier to set up and enforce.
7. Virtuozzo has true disk IOPS bandwidth management and limits, ensuring that tenants using too much disk IO have a very limited impact on other well-behaving tenants.

Conclusion

Without a doubt, Docker is a strong technology with amazing potential. It will likely change the way cloud applications are managed in the future.

Being a new and rapidly developing technology, Docker in the public cloud can greatly benefit from being deployed in a virtualized environment, and while hypervisor-based deployment is satisfactory, container virtualization with Virtuozzo can deliver much needed security and multi-tenancy requirements without the additional overhead that comes with using hypervisors.

Feature	Docker	Docker in KVM VM	Docker in Virtuozzo
CPU weights	Yes	Yes	Yes
CPU limits	--	Yes	Yes

Effective CPU overcommit	Yes	Limited	Yes
Effective NUMA management	User-space only	Yes	Yes
Effective memory overcommit	Yes	Limited	Yes
Network traffic management	--	Possible	Yes
Disk quota	Limited	Yes	Yes
Disk bandwidth management	--	Yes	Yes
Security isolation	--	Yes	Yes
Bare metal performance	Yes	--	Yes

Docker support in Virtuozzo allows service providers to build Docker-based services now, overcoming possible shortcomings of Docker for public cloud services. This integration delivers much required multi-tenancy and powerful resource management capabilities, and strengthens the security of the entire solution.

For More Information

Visit Virtuozzo: virtuozzo.com