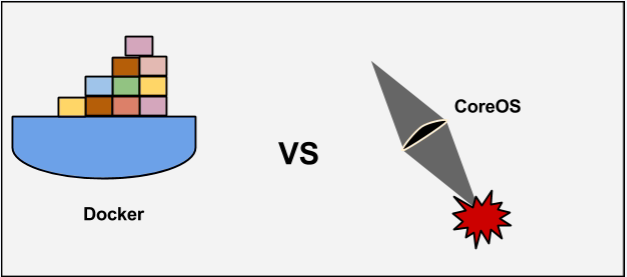
**Docker vs. rkt**



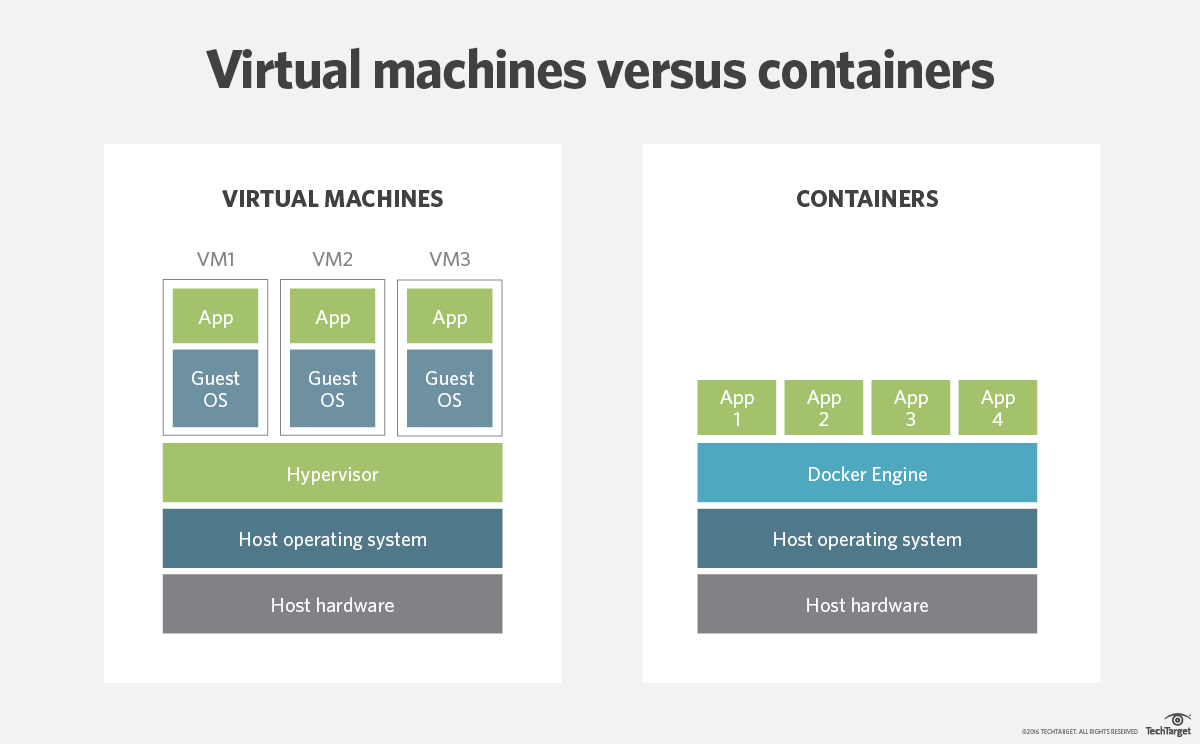
**Containerization** 🡪 also called container-based [virtualization](http://searchservervirtualization.techtarget.com/definition/virtualization) and application.

Containerization 🡪is an OS-level virtualization method for deploying and running distributed applications without launching an entire [VM](http://searchservervirtualization.techtarget.com/definition/virtual-machine) for each application. Instead, multiple isolated systems, called containers, are run on a single control host and access a single kernel.

Because containers share the same OS kernel as the host, containers can be more efficient than VMs, which require separate OS instances

Containers hold the components necessary to run the desired software, such as files, environment variables and libraries. The host OS also constrains the container's access to physical resources such as CPU and memory so a single container cannot consume all of a host's physical resources.

**Virtualization vs. Containerization**



**What is Docker?**

[Docker](https://github.com/docker/docker) is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package. By doing so, thanks to the container, the developer can rest assured that the application will run on any other Linux machine regardless of any customized settings that machine might have that could differ from the machine used for writing and testing the code.

In a way, Docker is a bit like a virtual machine. But unlike a virtual machine, rather than creating a whole virtual operating system, Docker allows applications to use the same Linux kernel as the system that they're running on and only requires applications be shipped with things not already running on the host computer. This gives a significant performance boost and reduces the size of the application.

And importantly, Docker is [open source](https://opensource.com/resources/what-open-source). This means that anyone can contribute to Docker and extend it to meet their own needs if they need additional features that aren't available out of the box.

**What is rkt (Rocket)?**

rkt is the next-generation container manager for Linux clusters. Designed for security, simplicity, and composability within modern cluster architectures, rkt discovers, verifies, fetches, and executes application containers with pluggable isolation. rkt can run the same container with varying degrees of protection, from lightweight, OS-level namespace and capabilities isolation to heavier, VM-level hardware virtualization.

[rkt is open source software](https://github.com/coreos/rkt/) written in the Go programming language

## **How is rkt different from Docker?**

* **Secure by Design**

In some systems such as **Docker**, if a user can break out of a container using a kernel exploit, the attacker can control the whole physical server and any attached data store. Of course, Docker can be secured to prevent a user breaking out, but that’s “additional” security, which requires time and effort.

In contrast,**rkt** runs containers as un-privileged users, so that even if a user breaks out, they cannot affect other containers or take control of the server. Further, **rkt** allows cryptographic signature checks on downloaded images so that only trusted containers can be run on the server.

* **Light Weight Design**

At the heart of Docker is a daemon process that is the starting point of everything Docker does. The docker executable is merely a REST client that requests the Docker daemon to do its work. Critics of Docker say this is not very Linux-like.

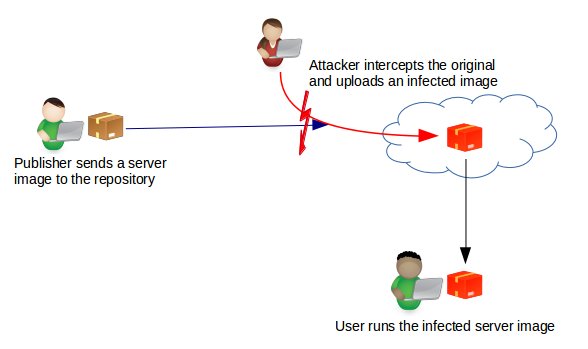
**rkt** just runs only those processes that are absolutely required by the app. In that aspect it is comparable to **Docker**, by minimizing the number of processes, resource utilization can be maximized in the servers, and it is easier to put all application dependencies into a small image and ship it to any server, anywhere.

* **Container image Security**

A great thing about Docker is that there’s a public registry from which anyone can download optimized application server images. So, if you want an Nginx server optimized for Magento web application, you’d get one from the Docker registry.

However, there’s a hidden danger in this. It is possible for an attacker to replace a server image with another one infected with malware.

Docker didn’t have a way to verify the authenticity of a server image. But in v1.8, a new feature called *Docker Content Trust* was introduced to automatically sign and verify the signature of a publisher.

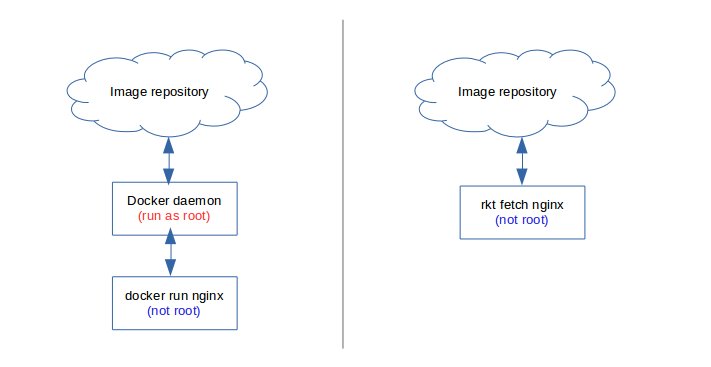


In **rkt**, signature verification is done by default. So, as soon as a server image is downloaded, it is cross checked with the signature of the publisher to see if it is tampered in any way.

### **Preventing “root” privilege escalation attacks**

**Docker** runs with super-user privileges. This Issue is a vulnerability in a container that can give an attacker root level access to the whole server. Docker always recommended running containers within SELinux or AppArmor, but many server owners consider it too complicated, and skip the step.

**rkt** came up with a better solution where new containers are never created from a root privileged process. In this way, even if a container break-out happens, the attacker cannot get root privileges.



As a way to contain this threat, Docker loads AppArmor security modules by default. This prevents one user (container) from seeing another user’s files or memory content.

In addition, many additional features like user name spaces, un-privileged execution, etc. can be configured to keep Docker containers secure.

### **Flexibility in publishing or sharing images**

When developing applications, it might be necessary to share container images with your technology partners. If you want to share Docker containers, you’ll need to setup a special private registry in your servers, or host it in a Docker paid account to share it with your partners.

For **rkt**, you just need your web server for it. **rkt** uses HTTPS protocol to download images and uses a meta description on the web server to point to the location. So, that’s one less server to maintain, and easier for partners to access.

* **Portability to other container systems**

In docker, migrating to a new container technology could be a problem, as it uses a proprietary image format.

In contrast, **rkt** uses an open source container format known as “appc”. So, any server image created using **rkt** can be easily ported to another container system as long as it follows the “appc” open format.

**Rkt** doesn’t enforce a vendor lock-in. This helps system owners migrate painlessly to another container system that suits their requirements better.

rkt allows you to use Application Container (appc) or Docker images. rkt supports the Docker image format and can interact with Docker repositories.

The nice thing about the appc format is that you are not tied to a particular repository like Docker. Instead, you have the possibility to host the image on a regular http-server or local file system. You can enrich the meta data in HTML files containing meta-tags.

**References:**

<http://searchservervirtualization.techtarget.com/definition/container-based-virtualization-operating-system-level-virtualization>

<https://coreos.com/rkt/>

<https://medium.com/@adriaandejonge/moving-from-docker-to-rkt-310dc9aec938#.567jbmbq3>

<https://bobcares.com/blog/docker-vs-rkt-rocket/1/>

<https://docs.docker.com/engine/getstarted/>

<https://hub.docker.com/>