# Cloud Computing Lab 7 – Review Week Word-count: 856

## Question 1:

“Describe how docker can be used to solve the increasing interoperability issues associated with a growing number of software products and versions used in the cloud services. Give specific Examples comparing Docker to Traditional Virtual Machines.

**Solving Interoperability:**

Interoperability is a massive issue for cloud services. This is due to the devices or software having dependencies or inter-dependencies where they won’t work with each other. Like modifying one piece of software might bring down the entire stack.

Isolation of software can be used to try and prevent inter-dependencies. Generally they’ll have a complex software system to ensure that all virtual machines aren’t dependent on each other.

However Docker is able to solve the problem of Interoperability by introducing standardised containers. Basically we can put whatever we want in the container and it’s sealed off from the rest of the software but we can take what we need out when we need it (in this case it’s generally libraries).

If we ship in containers, docker is a shipping container for sourcecode, we solve the Interoperability issue by giving a standard way of putting code into an interface, which we can ship around to where we want.

If we compare a Traditional Virtual Server to Docker we can see a number of very significant differences which makes Docker to be the better choice against most Traditional Virtual Servers…

**Traditional Server:**

* Has a host operating system
* Hypervisor to facilitate virtual instances on top of that
* Traditional hardware virtualisation, aims to create an entire virtual machine
* Each virtualised application includes not only the application - which may be only 10s of MB - and the necessary binaries and libraries, but also an entire guest operating system - which may weigh 10s of GB

To have multiple copies of that application running on a server you have to fully replicate the guest operating systems and the libraries associated with that, almost like running a full PC.

**Docker Server:**

Docker shares libraries and packages where appropriate

* The container comprises an application and its dependencies
* Containers serve to isolate processes which run in the solation in userspace on the host’s operating system.

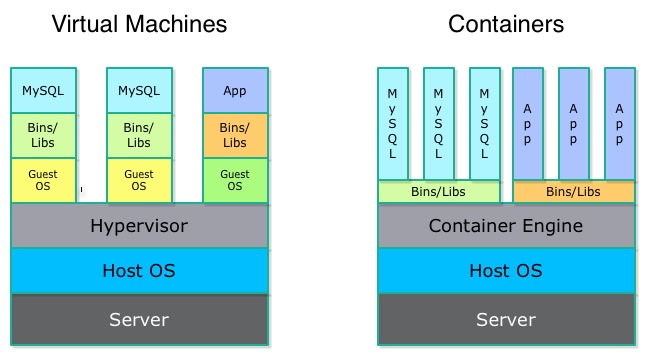
If we wanted to do that in Vmware, KVM and Zen virtual machines you’d have to create a full stack of all the software required for that virtualisation.

It massively increases the deployment size compared to docker due to the lack of reuse of software.

With a traditional VM, each application, each copy of an application and each slight modification of an application require creating an entirely new VM.

A new application on host need only have the application and its binaries/libraries.

They are basically shared in the same space to allow containers to work more efficiently.



**Automate Image Building:**

Docker supports automated, script-driven image generation

* A docker file specifics the following:
  + Base Linux image
  + Individual software installation commands
  + TCP/IP ports to expose
  + Default command(s) to execute when the container is run

**Docker vs Virtual Machine Summary:**

* ***Traditional VMs***
* Heavy and slow in comparison to docker system
* Takes full guest operating system for the basic requirement from which you build your application on
* ***Docker:***
  + Allows each container to share each library within the userspace so they be started and stopped very quickly and can be deployed much more efficiently

**Conclusion:**

**Overview of what Docker can do:**

* ***Run applications*:** An open source tool to run application inside of a *Linux container*, a kind of light-weight virtual machine
* ***Package applications***: In addition to running, it also offers tools to package containerised applications through *Docker Files*
* ***Distribute Application:*** Create your own *Docker registries or hubs,* a cloud service for sharing applications and automating workflows.

### Docker Architectural Overview:

* Docker uses a client-server architecture
* The *Docker client*talks to the *Docker daemon*, which does the heavy lifting of building, running and distributing your Docker containers
* Both the Docker client and the daemon can run on the same system, or you can connect a Docker client to a remote daemon.
* The Docker client and daemon communicate via sockets or through a RESTful API.

Docker drastically can reduce Interoperability; this is achieved through eliminating potentially redundant data such as including the OS on each machine rather than using the userspace. Furthermore the libraries are accessible through any of the other containers that need to use them.

Containers provide a way to isolate and easily manage the data by packing them away into separate locations, each container should be lightweight and shouldn’t contain massive amounts of data, this makes for a great deal of cohesion within the cloud network, reducing overall dependencies and workload on the machine due to there being no need to have all containers running at once. It is easy to make alternations to the containers as they are quick to start and stop and as mentioned should be isolated.

Overall Docker proves to be the better option is almost everyway due to its lightweight containerised system and this is clearly apparent as significant vendors such as IBM, Google, AWS, VMware, RedHat etc have announced support for docker.