# CS-512 Cloud Computing

Containerization
Submitted To:
Prof. Irfan
Submitted By:
Makaram Tayyab 2019-ag-6070
Ruman Waris 2019-ag-6082
M.Abubakr 2019-ag-6054

## Containerization:

- A form of virtualization  where applications run in isolated user spaces, called containers, while using the same shared operating system (OS).
- Container is a fully packaged and portable environment
- With containerization there is less overload during startup and no need to set up a separate guest os for each applications since they all share the same os kernel
- containerization is commonly used for packaging up the many individual microservices that make up modern apps.

## What is a container:

A container has the following things encapsulated in a isolated environment

- ❖ Binaries
- ❖ Libraries
- ❖ Configuration files
- ❖ Dependencies

Basically it has everything a application needs to run,the container is much like a lightweight virtual machine.

## How does Containerization work:

Each container is an executable package of software, running on top of a host OS. A host may support many containers (tens, hundreds, or even thousands)

This setup works because all containers run minimal, resource-isolated processes that others cannot access.

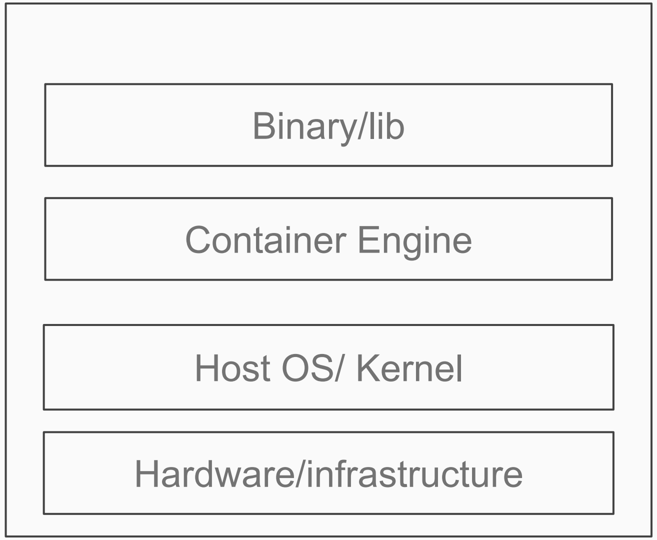
## Working of Containerization

Think of a containerized application as the top layer of a multi-tier cake:

- At the bottom, there's the hardware of the infrastructure in question, including its CPU(s), disk storage, and network interfaces.
- Above that is the host OS and its kernel—the latter serves as a bridge between the software of the OS and the hardware of the underlying system.
- The container engine and its minimal guest OS, which are particular to the containerization technology being used, sit atop the host OS.
- At the very top are the binaries and libraries (bins/libs) for each application and the apps themselves, running in their isolated user spaces (containers).

# Working of Containerization

Consider This Drawing as a example of containerization



## Evolution to Containerization

Containerization as we know it evolved from cgroups.cgroups is a feature in linux kernel it isolates and controls resource usage e.g how much CPU and RAM and how many threads a given process can access.Cgroups became Linux containers (LXC), with more advanced features for namespace isolation of components, such as routing tables and file systems. An LXC container can mount a file system, run commands as root, and obtain an IP address.LXC serves as the basis for Docker, which launched in 2013 and quickly became the most popular container technology—effectively an industry standard, although the specifications set by the Open Container Initiative (OCI) have since become central to containerization. Docker is a contributor to the OCI specs, which specify standards for the image formats and runtimes that container engines use.

## Benefits of containerization

The "lightweight" or portability characteristic of containers comes from their ability to share the host machine’s operating system kernel, negating the need for a separate operating system for each container and allowing the application to run the same on any infrastructure—bare metal, cloud—even within virtual machines (VMs), as we’ll see in the next section.
Similarly, developers can use the same tools when working with containers in one host environment as they’d use in another, which makes the development and deployment of containerized apps across operating systems much more simple.

## Containerization Technologies

There are a lot of containerization tools or technologies available some of them are listed below:

- Docker/Docker Enterprises
- CRI-O
- Rktlet
- Containered
- Microsoft Containers
- Kubernetes

## Containerization VS Virtualization

- Containers are typically measured by the megabyte. They don’t package anything bigger than an app and all the files necessary to run, and are often used to package single functions that perform specific tasks (known as a microservice). The lightweight nature of containers—and their shared operating system (OS)—makes them very easy to move across multiple environments.

- VMs are typically measured by the gigabyte. They usually contain their own OS, allowing them to perform multiple resource-intensive functions at once. The increased resources available to VMs allow them to abstract, split, duplicate, and emulate entire servers, OSs, desktops, databases, and networks.

# Containerization VS Virtualization

- Containers hold a microservice or app and everything it needs to run. Everything within a container is preserved on something called an image—a code-based file that includes all libraries and dependencies. These files can be thought of as a Linux distribution installation because the image comes with RPM packages, and configuration files. Because containers are so small, there are usually hundreds of them loosely coupled together—which is why container orchestration platforms (like Red Hat OpenShift and Kubernetes) are used to provision and manage them.
- Software called a hypervisor separates resources from their physical machines so they can be partitioned and dedicated to VMs. When a user issues a VM instruction that requires additional resources from the physical environment, the hypervisor relays the request to the physical system and caches the changes. VMs look and act like physical servers, which can multiply the drawbacks of application dependencies and large OS footprints—a footprint that's mostly not needed to run a single app or microservice.

## Drawworks of Containerization

- **The container ecosystem is fractured**. Although the core Docker platform is open source, some container products don't work with other ones — usually due to competition between the companies that back them. For example, OpenShift, Red Hat's container-as-a-service platform, only works with the Kubernetes orchestrator.
- **Persistent data storage is complicated**. By design, all of the data inside a container disappears forever when the container shuts down, unless you save it somewhere else first. There are ways to save data persistently in Docker, such as Docker Data Volumes, but this is arguably a challenge that still has yet to be addressed in a seamless way.

## Future of  Containerization:

- We foresee greater adoption. Containers are already highly penetrated in the organization. CNCF shows 60 to 70% deployment. But the percentage of the total computing workload running on Kubernetes (K8s) is much lower. As a result, there is a tremendous growth opportunity for K8s to get more of the workload.
- Containers will continue to disappear into the background like any good technology. Tools make it easier to leverage technology. There will be a greater simplification in the deployment and use of containers.
- Containers are making it easy for everyone to go serverless. There's no need to rely on a machine with a VM – that’s going away. It's easier to up and go serverless. Containers are going to improve over time. There will be more options to have more applications running inside of containers. They will continue to change, improve, become more stable, recover from failure more quickly, while returning big money savings.

## Research paper on Containerization in 2022

## Drawworks of Containerization

- **Graphical applications don't work well**. Docker was designed as a solution for deploying server applications that don't require a graphical interface. While there are some creative strategies (such as X11 video forwarding) that you can use to run a GUI app inside a container, these solutions are clunky at best.
- **Not all applications benefit from containers**. In general, only applications that are designed to run as a set of discreet microservices stand to gain the most from containers. Otherwise, Docker's only real benefit is that it can simplify application delivery by providing an easy packaging mechanism.