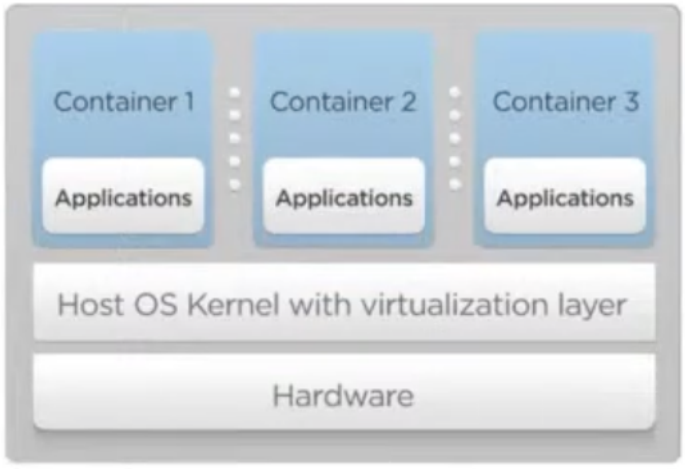
COMPSCI 677 Distributed and Operating Systems **Spring 2024**

Lecture 9: March 04

Lecturer: Prashant Shenoy *Scribe:* ***Sharvani Vempati(2024)***

9.1 Brief: OS Virtualization

OS virtualization uses the native OS interface to emulate another OS interface. A use case can be emulating an older version of OS. A popular use case is to allow backward compatibility, allow an older version of an application to be run, or sandboxing.



Questions: Is Windows Compatibility mode also an example of OS virtualization?
Answer: No, it does not use virtualization. It involves more of driver versions handling to pretend like an older version.

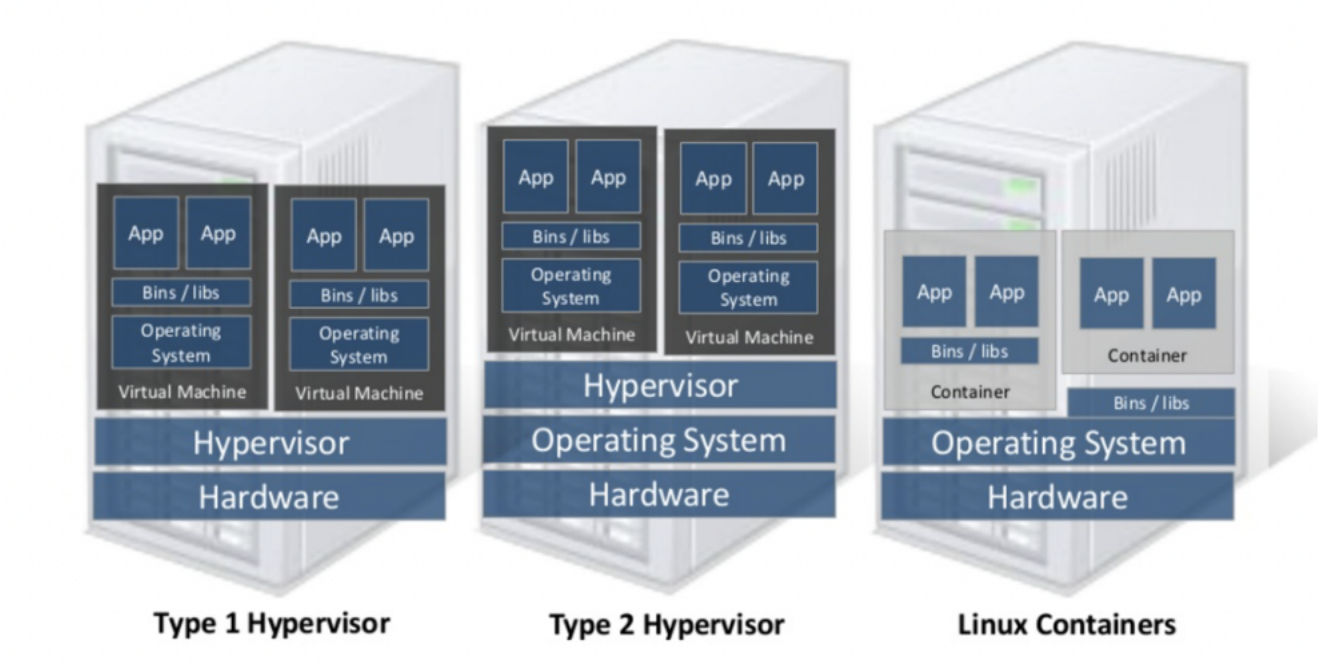
In OS virtualization, we create 'light-weight' virtual machines called containers. These containers are light-weight because they are built on top of the same underlying host OS. We cannot run an additional separate OS inside a container. Applications run directly inside containers and it uses the underlying host OS.

Question: Do we need to modify the host OS for OS virtualization?
Answer: The host OS needs to have support for OS virtualization, but it does not need any modification apart from that.

Question: What usecases does this have?
Answer: A base usecase for containerization is to islaote applications from one another. Docker is completely based on containerization.

9.1.1 Linux Containers(LXC)

Operating systems (OS) offer sandboxing and resource isolation capabilities. However, containers offer a more lightweight solution compared to virtual machines because they do not require an entire operating system to boot up. A container doesnot actually run an OS.



Question: In Linux containers do we have the ability to run multiple OS?
Answer: No we don't have the ability to run multiple OS on a linux container. In Linux containers we do not really run full pledged OS. We can emulate an OS interface inside the container.

9.1.2 OS mechanisms for Linux Container

9.1.2.1 Namespaces

Namespaces can allow control over what resources are visible to a process. Resources such as processes, mount points, network interfaces, users etc. Eg: If you restrict the processes that a container can see, if a user runs ps or top command on the container, it will only return a subset of the processes that are actually running on the physical machine.

Question: How is a sandbox different from a container?
Answer: The terms are being used interchangeably. Container is a technical term.

9.1.2.2 Cgroups

Cgroups or 'Container Groups' can allow control over how much resources a container can consume. We can control/limit the amount of CPU allocated, disk space allocated to a container. Eg: A container be

allocated 1 GB of memory. Thus, all processes running on the container can collectively consume only 1GB of memory and anything above that will start failing. Resources such as memory, CPU cores, I/O cycles, network bandwidth etc.

Question: Can resources allocated to a container be changed on the fly?
Answer: Yes, utilities available for that

9.2 Proportional Share Scheduling

This is a policy used to allocate resources across processes where-in each process is assigned a weight and it is allocated resources accordingly.This mechanism is widely used in virtualization to allocate resources to individual virtual machines (Type 1 and Type 2 Hypervisors) and containers. It is used to decide how much CPU weightage to allocate to each container and network bandwidth to each container. In share-based scheduling, a weight is allocated to each container and CPU time is divided in proportion to this weight. So if two containers have 1 and 2 as weights they will receive 1/3 and 2/3 of the CPU time. However, if the container is idle and is not using the resource, its share is redistributed across other containers/processes that have something to run in proportion to their weights (fair share scheduling).

Lottery-based scheduling: Fair share scheduling in a randomized way. We assign each container some number of lottery tickets. The CPU scheduler decides which process/ container gets the next time slot via a lottery – it's going to take all the tickets that have been assigned to all the containers and pick one winning ticket randomly. The container that holds the winning ticket gets to run next next time slice. The number of lottery tickets a container has determines the chance of winning the lottery. So, by controlling the number of lottery tickets, we can decide how much share of a resource a container will get.

Hard limits: Hard limit means there is a hard allocation. Even if the container is not using the resources, it is not redistributed across other containers. Those cycles are simply wasted.

9.2.1 Weighted Fair Queuing (WFQ)

Each container is assigned a weight w_i and it receives $w_i / \sum_j w_j$ fraction of CPU time. The scheduler keeps a counter for each container, s_i, which tracks how much CPU time each counter has received so far. The scheduler picks the container with minimum count so far and allocates a quantum time unit q. The counter value for the selected process is updated, $s_i = s_i + \frac{q}{w_i}$.

If one container is blocked on I/O and not using CPU, its counter value does not increase. However another container keeps on running and its counter value is incrementing. When the first container finishes I/O, it will have a low counter value as compared to second counter. Thus it will be scheduled for a long time while the second container is starved. This does not follow fair based scheduling. To avoid this the counter value is updated using this formula: $s_{min} = \min(s_1, s_2, \ldots)$ and $s_i = \max(s_{min}, s_i + \frac{q}{w_i})$.

Question: How does the scheduler know whether the container has used CPU cycles?

Answer: Let's only consider processes first. Each process gets a weight, and the share-based scheduler schedules these processes. These processes get CPU share in proportion to their weights. The CPU scheduler knows which processes are blocked on I/O and which processes are ready to run. If a process is blocked on I/O then it is not running and basically giving up its share. If some process is runnable, that means it is active. It will get to run and its counter will be incremented. It's the same concept for the container. There are processes in the container, so if any process from that container runs that container's counter is incremented based on the cycles used by that container. If there's nothing running, then the counter doesn't increment and something else gets to run.

Question: Is the number of processes static or dynamic?

Answer: The number of processes/containers are assumed to be dynamic. It will change over time. It means that the relative share of the resource you get will change over time. Say a container is given a weight of one. If there's nothing else in the system, it means that the container gets a hundred percent of the resource. If another container gets created and it also gets a weight of one, now the resource is shared 1:1. If a third container comes now again the allocated fractions will change.

Question: In practice who is managing the weight?

Answer: The user manages the weight that is given. If nothing is given it just assumes a weight of 1.

Question: Why is $s_i = \max(s_{min}, s_i + \frac{q}{w_i})$?

Answer: This says that if a process P becomes inactive and then comes back and becomes active, its counter has to be at least the min counter of the system. This is because its counter was the same for a while and so it's likely to be the least, as everyone else's counter has advanced. Now, if I simply pick the min=1, P will start hogging the resources. So, whenever a process becomes active its counter has to be at least the minimum counter in the system, and then from that point on, its counter is incremented. If a new process arrives and there are existing processes running its counter can't be initialized to 0. Its counter has to be initialized to the min counter of any active task in the system. That way it is at least starting as the minimum task in the system and not 0.

Question: Is the value of s_{min} going to always increase?

Answer: Yes. It's basically tracking the process/container that has received the least service (least amount of cycles in the system). So, it will increase monotonically.

9.3 Docker and Linux Containers

Docker uses abstraction of Linux containers and additional tools for easy management.

1. Portable Containers - With LXC, we would have to use namespaces and cgroup commands to construct a container on a machine. With Docker, all of this information can be saved in the container image and this image can be downloaded and run on a different machine.
2. Application Centric - Docker can be used for designing applications quickly. Software can be distributed through containers.
3. Automatic Builds.
4. Component Reuse - Helps to create efficient images of containers by only including libraries/files not present in the underlying OS. Achieved using UnionFS.

Question: Does Docker deal with packages dependencies for the applications running in the container?
Answer: Packages needed for an application are included in the Docker image. Specific version libraries that are required for application need to be included in the image if they are not present in native OS. OS Does not need to worry about these libraries or issue of incompatible versions.

Question: Docker images are also available for Windows/MAC. How is it possible to run Linux Containers on Windows/Mac?
Answer: Docker provides a hidden VM of linux and containers run on top of it. These are small barebone linux kernels that run on non-Linux platform. Docker does not use true OS virtualization for running Linux

containers on other platforms, as it would have to translate Linux calls to other platform calls (Windows/Mac Calls).

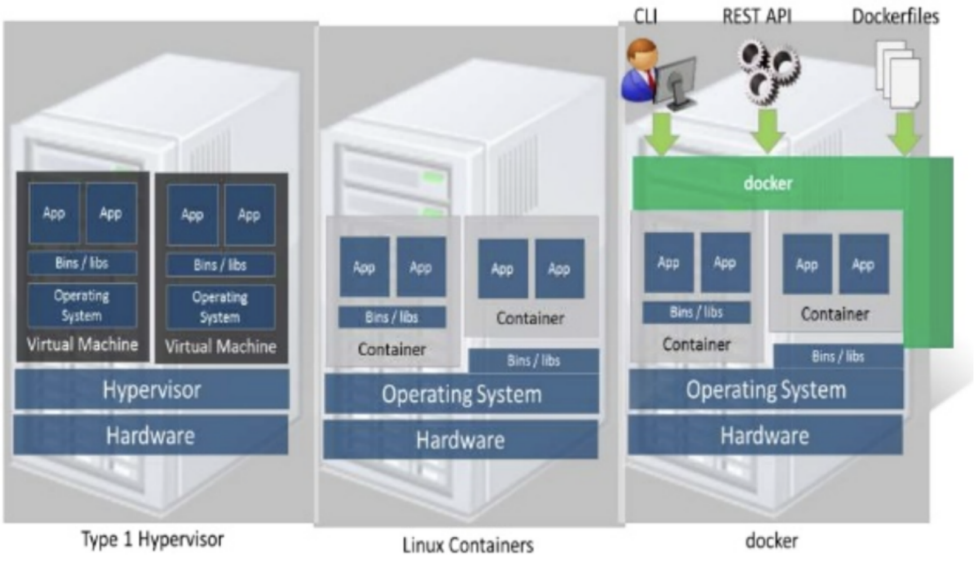


Figure 9.1: Visualization of type 1 hypervisors vs Linux containers vs docker.

Docker is basically a layer around the Linux container (refer Fig. 9.1) that provides tools to build these containers very easily and distribute them. For example, we can distribute them through git repositories. We can just download a container from a git repository and run it on any machine. Docker images will run anywhere Docker runtime is present. All we need to do is download a Docker image and just say run. We don't have to create a container or configure it (e.g., allocate resources). Docker has done all of it.

Docker is a Linux-based OS virtualization technology. However, it can run on any platform (e.g., Mac OS). This is because there is a hidden version of Linux that Docker is starting which is not visible to us.

Question: Is Docker a type 2 hypervisor?

Answer: Docker is a container management runtime. So, it is not technically a type 2 hypervisor. It is using a type 2 hypervisor to run Linux in the background, and then it's running on that Linux.

Question: Is Docker performance the same in all kinds of systems?

Answer: If we are running Docker in any virtualized environment, there will be some overhead which is not going to be present if we are running it in a native environment. For example, if Docker is running on native Linux, the overhead will be less than if it is running on a Linux VM that's running on a Mac.

Question: If you have multiple containers running, do each of them run on a separate VM?

Answer: All of them run on a single Linux instance. if the machine does not have native Linux it will create a virtual machine with Native Linux and all the containers will run on that machine.

Question: If you create a virtual Linux can you go into the shell of the Linux?

Answer: The docker prompt that was visible during the demo was a shell on that Linux machine. It's just that it's inside a container.

Question: To create a python image, you cannot enter the shell; but if you do Ubuntu, you can enter.

Answer: That's not the case. All containers, regardless of what is packaged in them (say, python package), run on a Linux instance. So, we will get a shell, which is a shell for the container. The application is sandboxed in the container.

Question: In the language image, we cannot enter the shell.

Answer: That's not the case. Whatever the language runtime is, it is still running on that virtualized Linux instance. Even though we showed a web server package in the demo, we still haa a shell that is on the native instance. If we do "top", we will see our process(es) running. The Linux instance may have other running processes, but we don't see everything as we did a namespace and hence we can see only the processes that are running inside our container.

Question: Are namespace and containers the same?

Answer: No. A namespace is a way by which we can limit what a process can see. A container is a namespace + C-group (which allocates some resources to those processes) + some other things. A container has a namespace associated with it, but a namespace itself doe not make a container.

9.3.1 Docker Images and use

Docker uses a special type of file system called as a union file system (AuFS). Docker assumes that there are a base set of files that are already present on the Linux OS and if the container needs the same files then the existing set of files on the OS are used. If they are not already present the files are bundled with the docker image. The container images are made compact due to this process.

PlanetLab: Virtualized architecture used for research by students in different locations.