CMPSCI 677 Operating Systems **Spring 2022**

Lecture 9: February 23

Lecturer: Prashant Shenoy *Scribe: Samriddhi Raj (2022), Xingda Chen (2019)*

9.1 OS Level Virtualization

OS level virtualization is the use of the native OS interface to emulate another OS interface to mimic system calls. Lightweight virtual machines have no hypervisor. There exist containers that are light weight VMs which are constructed using OS level virtualization. This is a virtual machine that allows user to take resources from physical machine and construct containers. Each container can run one or more applications. All containers are managed through native operating systems. Difference between OS and hardware level virtualization including the following: hardware level virtualization has the abstraction of the hardware, which is emulated by the virtualization layer. You first have to run the OS in the VM and then the application runs in the OS. In OS level virtualization, when there are containers, the OS doesn't run in VMs. The OS is the same as what it runs on the actual machine, the containers runs on the native OS. Applications run using OS level virtualization are "sand-boxed", which means that don't see other running applications and they only see the resources allocated in their only containers. Containers are an isolated set of processes.

Benefits of putting applications into containers include:

1. : Isolation properties, better CPU usage allocation
2. : Security, limiting which files an application can see
3. : Good way to distribute software

Question: Why is creation and start-up of containers much faster than virtual machines ?
Answer: To start a container we just need to start an abstraction of operating system and put restrictions in them as to what the processes inside can access and see. Whereas to start a VM, we need to boot the entire virtual operating system. Thus, containers have faster provisioning due to less overhead during the OS boot.

Question: Does a malicious application in a container corrupt the host OS?
Answer: Yes, a malicious application can corrupt the host OS. In hardware level virtualization is this not the case? The OS is running inside the VM, if one VM is down the other VM continues to run. OS virtualization has less security than hardware virtualization? Every VM gets its own OS.

Question: In Docker how we can sometimes use different operating system like an image inside the contianer?
Answer: Docker uses linux containers. But the entire kernel is not booted up inside there. We can use application packages of a different OS distribution in there so that it looks like this corresponding OS environment. These application files exposed inside the container makes it look like a specific distribution. But there is only one underlying operating system being used, for example, Ubuntu.

Linux containers are lighter weight compared to hypervisors. Provisioning is fast due to a simpler structure. The container can emulation different OS interfaces. There is only one scheduler which resides in the OS. The scheduler in the OS ensures each container only gets its assigned CPU time.

9.2 OS Mechanisms for LXC

Container Abstraction is designed using two methods:

1. Namespace - Restrict what the applications/processes inside a container can see.
2. CGroups (Container Groups) - Restrict what the applications/processes can use. For example: CPU/RAM Usage (Resource Isolation).

Namespace and CGroups are independent abstractions that allow the implementation of the full container abstraction. In Linux both are designed as independent abstractions.

Additional Abstractions on top of baseline container abstractions include:

1. chroot: changes the root directory to a user specified directory.
2. Docker offers SDE tools that allow to create and distribute images as containers.

9.2.1 Namespaces

Namespaces restrict what a container can see. Some of namespace restrictions are:

1. Which other processes may be seen. For example, 3 Processes inside a container can see each other. They cannot see other processes running on other containers or on underlying OS.
2. Abstract Process IDs of other processes.
3. Restrict File Systems/ Directories/ Mount Points - Storage Restrictions
4. Network Detail Restrictions
5. User ID Visibility

Example: A machine has total 10 user accounts. Only 2 user accounts are visible on a container. The other 8 users cannot access this container.

9.2.2 Linux Cgroups

CGroups puts an upper limit to the amount of resources a container can use. Example - A machine has 16GB RAM and 4 cores, but the container can also use 1GB RAM and 1 core respectively. Specialized scheduler like weighted proportional share scheduler can be used to allocate limited resources to a container.It allocates a weight to a container and CPU time is allocated to the containers in proportion to the weight. In the situation when the container is not utilizing the assigned CPU time, hard allocation will just waste the CPU time and fair resource distribution will redistribution the cycle accordingly, it is also called work concerning scheduler. A different technique known as Core Pinning can be used to assign container to a particular core. CPU Scheduler/OS will not run processes belonging to this container on other cores.

9.3 Proportional Share Scheduling

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.

Hard limits: Hard limit means there is a hard allocation,no matter if a process uses it or not, it is allocated.

9.3.1 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 $s_i = \max(s_{min}, s_i + \frac{q}{w_i})$.

9.4 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 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 container on other platforms as it would have to translate linux calls to other platform calls (Windows/MAC Calls).

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

9.5 Code, Process and VM Migration

The motivation behind developing techniques for code, process and VM migration is that migrating these components of a system helps improve performance and flexibility.

There are two types of migration models:

- **Process Migration:** Also known as strong mobility, this includes the migration of all the components of a process, i.e., code segments, resource segments and execution segments. An active process (an already executing program) on a machine is suspended, its resources like memory contents and register contents are migrated over to the new machine and then the process execution is restarted. It involves significant amount of data transfer over the network.
- **Code Migration:** Also known as weak mobility. In this model only the code is migrated and the process is restarted from the initial state on the destination machine. The network transfer overhead is low since only the code is transferred. Some examples of code migration are web-form, Flash/Java applets in browser, web search. Docker is also considered to be an example of code migration.

Question: Why is a search query an example of code migration?
Answer: Keywords typed in a search bar basically become part of a query and query is a program. On pressing submit, the query is sent to another machine and is executed there. This illustrates migration of code from client machine to the server machine.

Question: In process migration, if you suspended an active process and migrated it, how do you take care of its state?
Answer: The specific state of process like its memory contents can simply be written onto a disk and the process can be resumed elsewhere. Debuggers perform a similar operation.

Question: Does the migration have to be only between client and server or can it be between a cluster of servers?
Answer: Migration is not limited to just client and server. Migration is independent of the source and destination of the code or process.

An example of sender-initiated migration is a web search and database query. An example of receiver-initiated migration is a browser downloading a Java applet or Flash application from the server.

A process can be migrated or cloned. In case of migration, the complete process is moved to a different machine. In cloning, a copy of the process is created on a different machine and you allow both the copies to execute. Cloning is a convenient way of replicating the process. An example of cloning is forking a process.

To decide whether to migrate a resource attached to a process or not, we look at the nature of binding of resource to process. There are three types of resource to process bindings:

- **Identifier:** Hard binding, one that you cannot substitute. Least flexibility. Example is URL for a website
- **Value:** Slightly weaker binding. Libraries used in Java are a good example.

- **Type:** Weakest binding, one that can be substituted. Maximum flexibility. Example is a local device like printer.

Apart from the resource binding, it is also necessary to look at the cost of moving resources which can also be classified into three categories:

- **Unattached:** Very low cost of moving. Example- files.
- **Fastened:** High low cost of moving. Example- databases.
- **Fixed:** Can't be moved. Example- local devices.

Different combinations of resource-to-machine binding and process-to-resource binding are tabulated below:

	Unattached	**Fastened**	**Fixed**
By Identifier	MV (or GR)	GR (or MV)	GR
By Value	CP (or MV, GR)	GR (or CP)	GR
By Type	RB (or GR, CP)	RB (or GR, CP)	RB (or GR)

where GR means establishing global system-wide references, MV means moving the resources, CP means copying the resource and RB means rebinding process to locally available resource.

Question: Would you want to do checkpoint and restart instead of migrating a process?
Answer: Checkpoint and restart is a standard way to implement process-migration and not an alternative to process-migration.

Question: Would you want to do checkpoint and restart instead of migrating a process?
Answer: Checkpoint and restart is a standard way to implement process-migration and not an alternative to process-migration.

Question: If JVM is a value resource, why can't we bind JVM of the new machine to newly migrated process ?
Answer: If exact same version of JVM as required by the process is available on the new machine, then we can bind it to the process. However a different JVM version can be incompatible and cause the program to crash.