## Lecture 1

#### Virtualization

Multiple virtual machines (VMs) can run inside a physical machine (PM). A user of the VM is given the illusion that they have the full harware to themselves (but in reality many VMs are sharing the same underlying hardware).Containers is another virtualization technology, they are like lightweight VMs.

Virtualization is a building block for cloud computing. Virtualization enables multiple clients share the cloud’s compute resources. Multiple users on VMs/containers can share the same cloud server. Without virtualization, the cloud wouldn’t be a profitable idea.

#### Why cloud computing?

Bare-metal refers to a computer system or server that is used without any virtualization layer pre-installed. In other words, the term "bare-metal" implies that the hardware is directly accessible and utilized without the abstraction layers that virtual machines or containerization technologies provide.

Why run applications on cloud and not on bare metal servers?

* Multiplexing gains: multiple VMs can share the system resources. It stems from the fact that the machines are not utilized at 100% capacity by each user; there’s some spare capacity that can be shared.
* Lower overhead of maintenance: hardware/software maintained by providers (if using public cloud).
* Flexibility: VMs can move to another physical machine if one fails.
* Pay as per usage: no need to invest in servers if only lightly used.

Disadvantages of running applications on cloud:

* Performance: longer delay to access server via internet.
* Higher cost if heavily used

#### Virtualization terminology

System virtualization: running one full system (OS and applications) over another OS.

Process virtualization: gives a single process the illusion that it is running on a different architecture; it lets a single process run on a different architecture from the underlying machine.

Hypervisor or Virtual Machine Monitor (VMM): a piece of software that allows multiple VMs to run on a PM.

A group of blue and orange rectangular boxes with text

Description automatically generated with medium confidenceThe hypervisor allows the VMs to share the underlying PM.

The Guest OS runs inside the VM, and the Host OS runs on the PM.

**Type 1 Hypervisor**: runs directly on hardware (no need for host OS); it takes on the role of the host OS as well. It is VMM plus an OS built-in together.

**A close-up of several types of computer hardware

Description automatically generatedType 2 Hypervisor**: runs as an application on top of the host OS; it does not have an OS functionality.

#### Challenges to virtualization

A group of blue and orange rectangular boxes with text

Description automatically generatedThe Guest OS expects complete control over hardware, but the VMM must multiplex multiple guests on the same hardware.

Some ways to design VMMs:

* Hardware assisted virtualization (e.g. KVM/QEMU): modern CPUs have support for virtualization and VMMs are built over this support
* Full virtualization (e.g. VMWare): original technique to run unmodified OS (as Guest OS) over original hardware with no virtualization support (the hardware provides no support for virtualization).
* Paravirtualization (e.g. Xen): OS source code is modified to be compatible with virtualization.

When the underlying hardware does not support virtualization, we have to use the full virtualization technique in order to run OSs as Guest Oss.

The virtualization applies to all hardware resources: CPU, memory, I/O devices, etc (they all need to be shared). All of them need to be virtualized.

## Lecture 2

#### Process

A process is a running program.

A program is compiled to generate an executable. The executable contains machine/CPU instructions. The compiler translates high level language code to instructions the CPU can run.

When an operating system creates a process, it allocates memory in the RAM for the process's memory image. This memory image is a structured representation of everything the process needs to run. The **memory image** includes the code and static/global data extracted from the executable file. This section contains the instructions the CPU will execute and any data that is statically allocated or global within the program. Additionally, the operating system allocates heap memory for the process – some amount of extra memory is allocated for the process in the form of heap in order to handle dynamic memory allocation (that is, runtime memory allocation). The heap is used for dynamic memory allocations, such as those performed by functions like *malloc* in C, allowing the process to request and manage memory during its execution.It also included the stack, which is used to store information necessary for function calls, including the arguments passed to functions, the return addresses (where the function should return control after execution), and the local variables. This organization ensures that each function has its own space to operate without interfering with others, allowing for proper execution and return of functions within the process. Through these allocations, the operating system ensures that the process has all the necessary memory segments to execute its code, manage dynamic memory, and handle function calls effectively.

All instructions and variables in the memory image are assigned memory addresses. Starting at 0, up to some max value (4GB in 32-bit systems).

Memory image of a process: A screenshot of a computer

Description automatically generated

When a process is run, the CPU executes the code in the memory image.

When a process runs on the CPU, the CPU registers hold values related to the process execution. The program counter has the address of the current instruction. The CPU fetches the current instruction, decodes it, and executes it. Any variables needed for operations are loaded from process memory into general purpose CPU registers (the add instruction for example requires to fetch some value from memory, store it into some register and them perform the addition these register values). . After the instruction completes, the values are stored from registers into memory. The stack pointer has the address of the top of the stack (the current stack frame holds arguments/variables of the current function that is running).

The set of values of all CPU registers pertaining to a process execution is called its CPU context.

8 min

<https://www.youtube.com/watch?v=ix9Ylli70yY>