UniTs - University of Trieste

Faculty of Scientific and Data Intensive Computing

Department of mathematics informatics and geosciences

Cloud Computing

Lecturer:
Prof. Taffoni Giuliano

Author:
Christian Faccio

March 25, 2025

github.com/christianfaccio christianfaccio@outlook.it

Contents

1 Introduction

Definition: ***What is Computing?***

Computing is the process of using computer technology to complete a given goal-oriented task. Computing may encompass the design and development of software and hardware systems for a broad range of purposes.

Today, each scientific instrument is critically dependent on computing for sensor control, data processing, international collaboration, and access. Computational modelling and data analytics are applicable to all areas of science and engineering. The ability to use computing effectively is a key skill for all scientists and engineers.

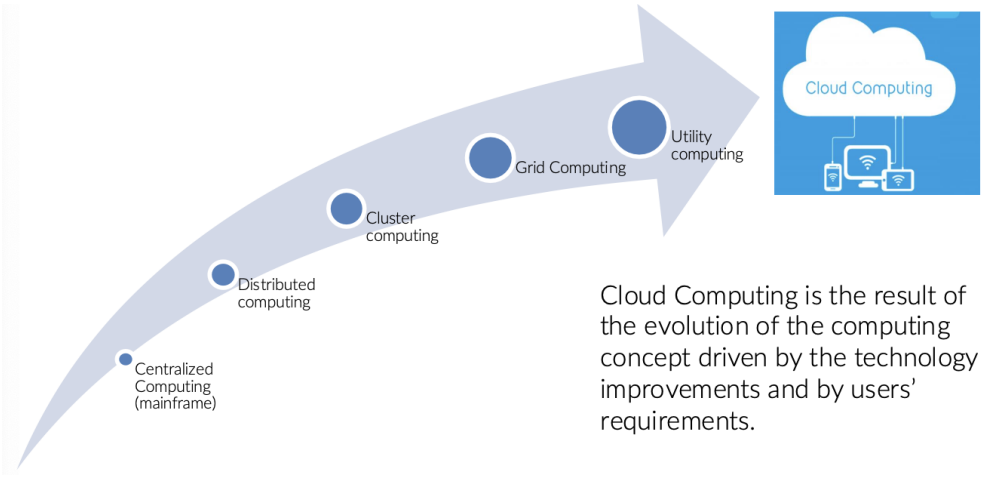


Figure 1.1: The evolution of computing.

We can now define what **Distributed Computing** is:

Definition: ***Distributed Computing***

A distributed system is a collection of autonomous computers that are interconnected with each other and cooperate, thereby sharing resources such as printers and databases.

A **DD Architecture** is a distributed system that consists of multiple autonomous computers that communicate through a computer network. The computers interact with each other in order to achieve a common goal. It is based on a client-server model, where the client requests services from the server.

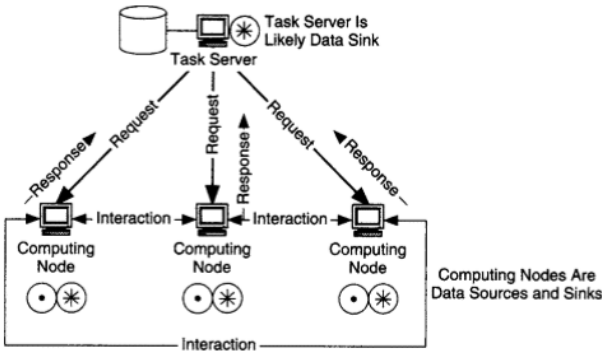


Figure 1.2: Distributed Computing.

There is also a 3-tier architecture, where the client interacts with the server, which interacts with the database.

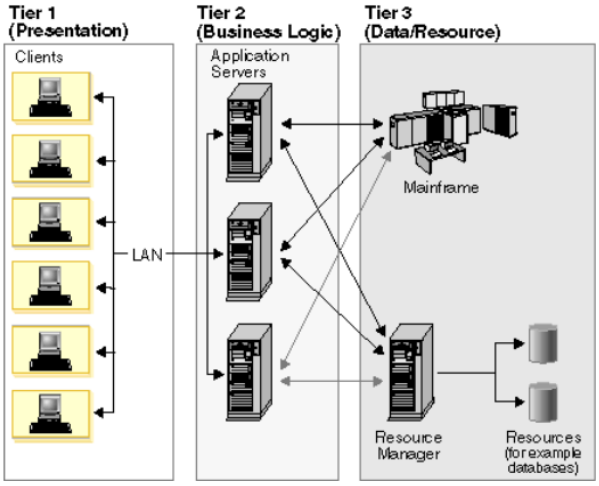


Figure 1.3: 3-tier architecture.

And finally a peer-to-peer architecture, where each computer can act as a client or a server. Responsibilities are uniformly divided among all machines.

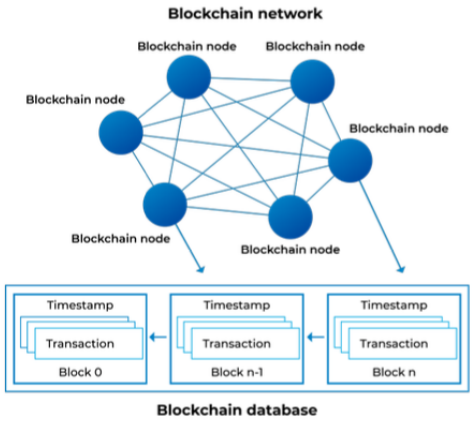


Figure 1.4: Peer-to-peer architecture.

Example: *Hadoop*

Hadoop is an open-source software framework for storing data and running applications on clusters of commodity hardware. It provides massive storage for any kind of data, enormous processing power and the ability to handle virtually limitless concurrent tasks or jobs. It implements a distributed scalable computing model for data analytics, using a **Distributed File System** (HDFS) and a **MapReduce** programming model.

- **HDFS**: Hadoop Distributed File System, a distributed file system that provides high-throughput access to application data. It manages a large number of large files, distributing them across the nodes in a cluster.
- **MapReduce**: a programming model for processing and generating large data sets with a parallel, distributed algorithm on a cluster.

A **Computer Cluster** is a group of linked computers, working together closely so that in many respects they form a single computer. The components of a cluster are commonly, but not always, connected to each other through fast local area networks. Clusters are usually deployed to improve performance and availability over that of a single computer, while typically being much more cost-effective than single computers of comparable speed or availability. Clusters are usually deployed to improve performance and/or availability over that provided by a single computer, while typically being much more cost-effective than single computers of comparable speed or availability.

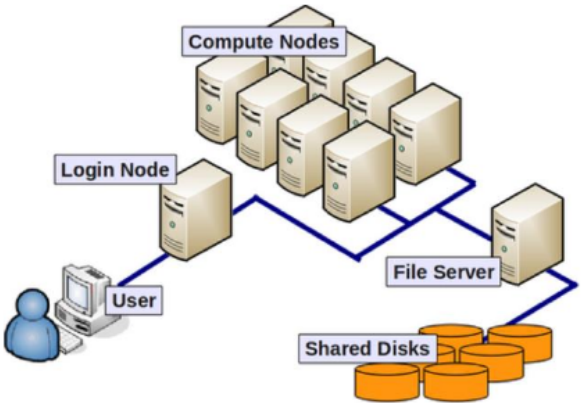


Figure 1.5: Computer Cluster.

A cluster can be accessed through a batch system, which is a software system used to manage and schedule batch jobs. Batch systems are used in environments where users do not interactively use a computer system, but instead enter a set of commands to be executed by the system at a later time. The filesystem structure is shared among all nodes in the cluster.

High Availability Cluster (Linux)	Network Load Balancing Cluster	HPC Cluster
Mission-critical applications	Operate by distributing a workload evenly over multiple backend nodes	Low-latency network
High-availability clusters (aka Failover Clusters) are implemented for the purpose of improving the availability of services which the cluster provides	Typically, the cluster will be configured with multiple redundant load-balancing front ends	Message-passing libraries
Provide redundancy	All available servers process requests	Parallel filesystem
Eliminate single points of failure	Web servers, mail servers, etc.	HPC system software

Table 1.1: Classification of Clusters

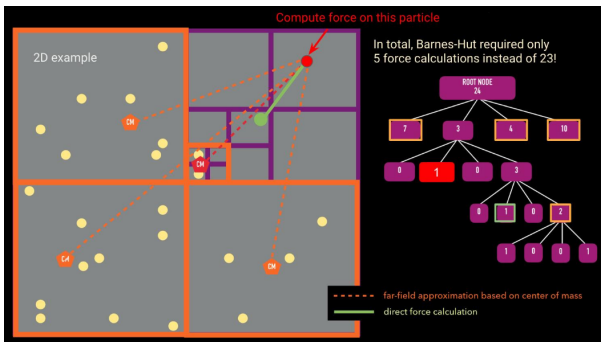
Observation: *HPC vs HTC*

High-Performance Computing (HPC) is the use of parallel processing for running advanced application programs efficiently, reliably and quickly. The term applies especially to systems that function above a teraflop or 10^{12} floating-point operations per second. The term **High-Throughput Computing** (HTC) refers to the use of many computing resources over long periods of time to accomplish a computational task.

- The **computing challenge**: In order to to improve the codes' performance, multi (multi–core CPUs) and many cores (GPUs) architectures have to be exploited.
- The **memory challenge**: Huge datasets cannot be loaded in the memory of a single CPU and cannot be handled by a single processor but by distributed memory systems. Distributed computing, based on the adoption of the MPI standard, represents a feasible and effective solution.
- **The data challenge**: This addresses the management, archiving and access of the raw data, the science data products, and the final outcomes of data processing and analysis.

N-BODY PROBLEM

The N-body problem broadly describes the problem of predicting the future trajectories of a group of objects under the mutual gravitational forces they exert on one another, given each individual object's current position and velocity. In Astronomy, the N-body problem has been studied at a wide variety of scales: ranging from the study of asteroids near Jupiter (Brož et al. 2008) to the study of the largest gravitationally bound clusters in the Universe (Angulo et al. 2012).



Including the communication overheads, a simple parallel algorithm has the following steps:

1. Build the Octree (on primary MPI node), Octree is broadcast to all MPI nodes.
2. For each particle, compute the total force by traversing the Octree (particle positions and velocities are distributed across MPI nodes).
3. Update the velocities and positions of the particles.
4. Repeat steps 2 and 3 for a number of time steps.
5. Output the final positions and velocities of the particles.

Definition: *Grid Computing*

Grid computing is the collection of computer resources from multiple locations to reach a common goal. The grid can be thought of as a distributed system with non-interactive workloads that involve a large number of files. Grid computing is distinguished from conventional high-performance computing systems such as cluster computing in that grid computers have each node set to perform a different task/application.

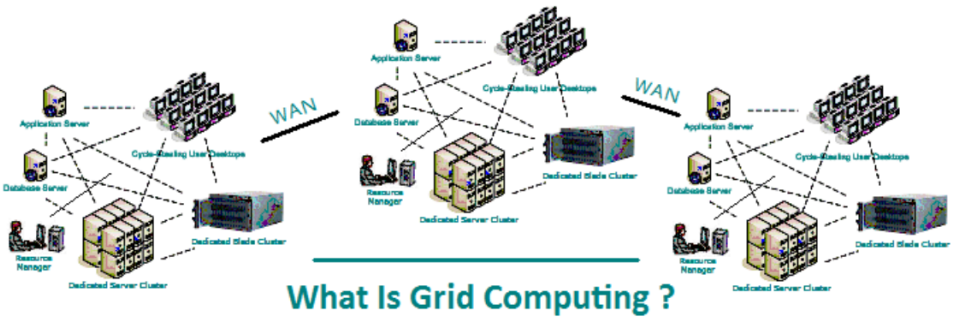
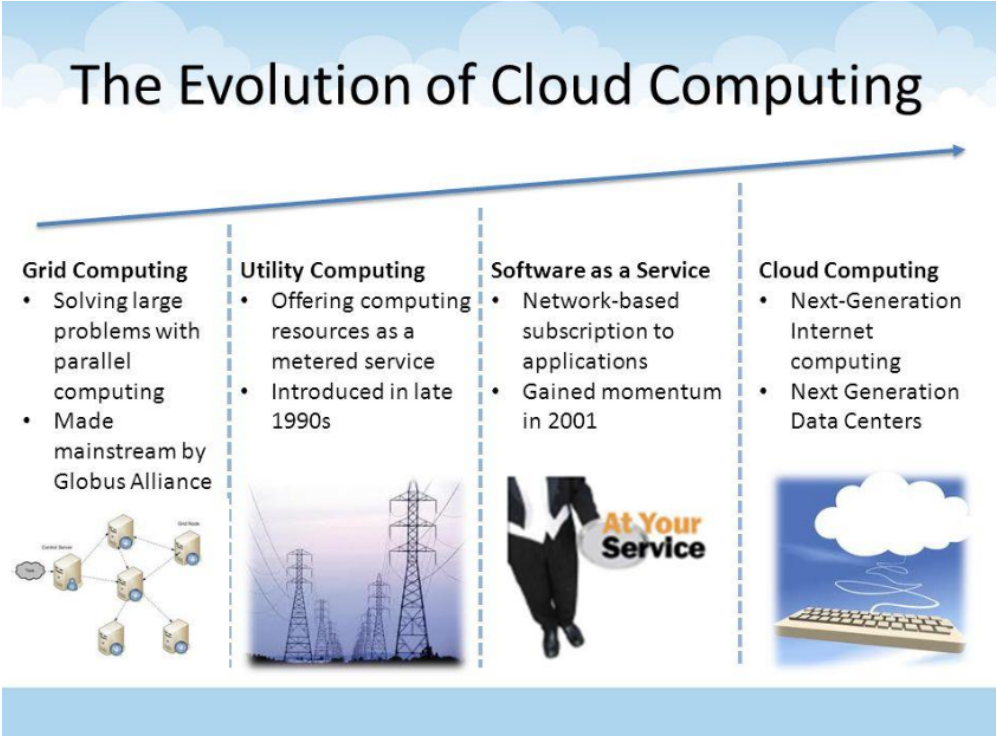


Figure 1.6: Grid Computing.

Utility computing is a theoretical concept used in Cloud Computing to provide services based on a metered service model. This model has the advantage of a low or no initial cost to acquire the service; instead, computational resources are essentially rented. The concept is analogous to other utilities, like water and electricity, where the consumer pays for what they use.

- Pay-per-use model
- Optimize resource utilization
- Outsourcing
- ”Infinite” resources
- Access to applications or libraries
- Automation

The principle of utility computing is very simple: One company pays another company for servicing. The services include software rental, data storage space, use of applications or access to computer processing power. It all depends on what the client wants and what the company can offer. Different model may be implemented even if the pay per use is the most common one (e.g. flat rate, metered, etc). The pricing model is what characterize the Utility Computing.



Edge Computing is a distributed computing paradigm in which processing and computation are performed mainly on classified device nodes known as smart devices or edge devices as opposed to processed in a centralized cloud environment or data centers. It helps to provide server resources, data analysis, and artificial intelligence to data collection sources and cyber- physical sources like smart sensors and actuators. A network of micro data centers embedded in the instruments/sensors that store or process critical data locally and push received data to a centralized data center or repository of cloud storage. Edge computing processes the data locally results in reduced traffic in the central repository.

- Computing at the edge of the network
- Focuses on bringing computing as close to the data source as possible
- It decentralizes processing power
- It reduces latency
- It improves data security
- It reduces the amount of data that needs to be moved
- It improves scalability

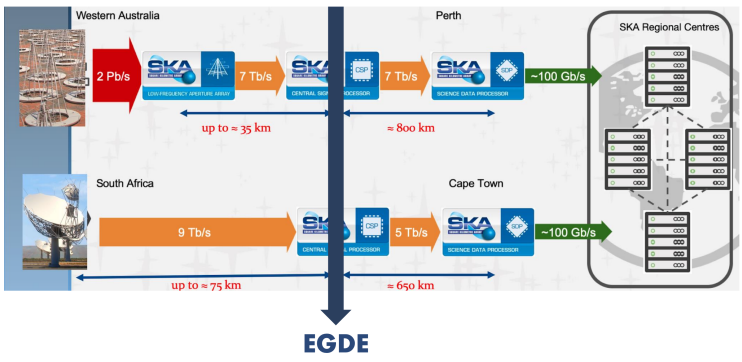


Figure 1.7: Edge Computing.

2

Cloud Computing concept and architecture

Definition: *Cloud Computing*

Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential characteristics, three service models, and four deployment models.

Cloud computing is the on-demand availability of computer system resources, especially data storage (“cloud storage”) and computing power, without direct active management by the user. Clouds may be limited to a single organization (private/enterprise cloud), or be available to many organizations (`public` cloud), maybe a mixture of the two (`hybrid` cloud); Cloud implements a `pay-as-you-go` model based on the concept of infinite resources availability;
Base concepts:

- **Abstraction**: the process of removing or reducing the complexity of a system by hiding or suppressing details;
- **Virtualization**: the process of creating a virtual version of something, including virtual computer hardware platforms, storage devices, and computer network resources;

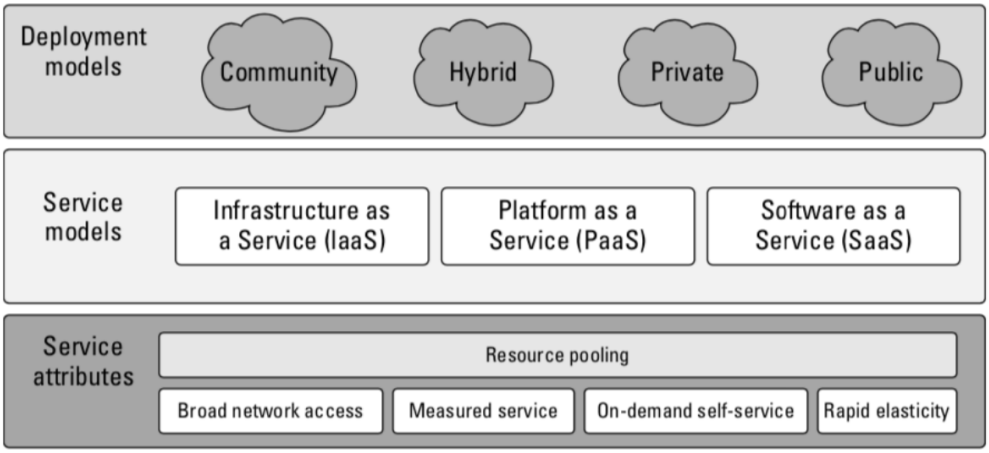


Figure 2.1: Cloud Computing Architecture

2.1 Service Attributes

- **On-demand** On-demand computing is a business computing model in which computing resources are made available to the user on an ”as needed” basis. Rather than all at once, on-demand computing allows cloud hosting companies to provide their clients with access to computing

resources as they become necessary. The on-demand computing model overcomes the common challenge that enterprises encountered of not being able to meet unpredictable, fluctuating computing demands efficiently.

- **Broad network access** Capabilities are available over the network and accessible through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations).

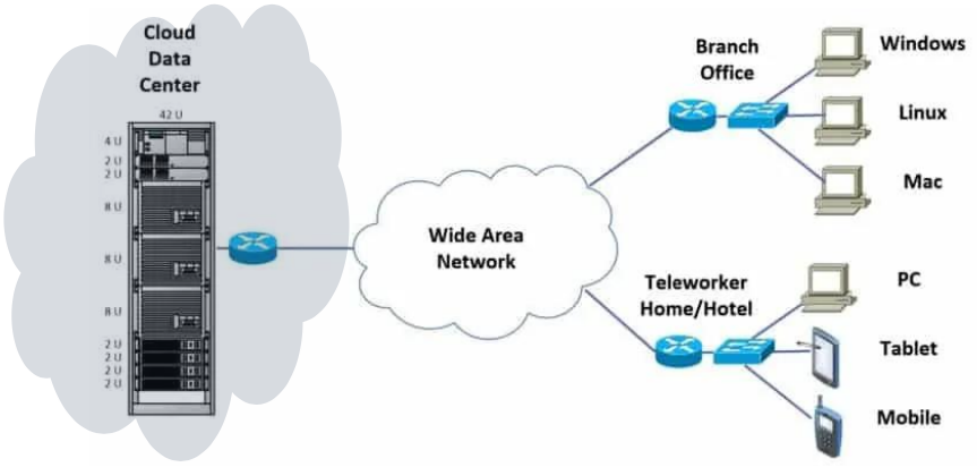


Figure 2.2: Broad network access

- **Resource pooling** Computing resources are storage, processing, memory, network bandwidth and virtual machines. Provider's computing resources are pooled to serve multiple consumers, allocated and deallocated as needed. Tenants are Isolated. Location independence: there is no control over the exact location of the resources. This has major implications performance, scalability, security.

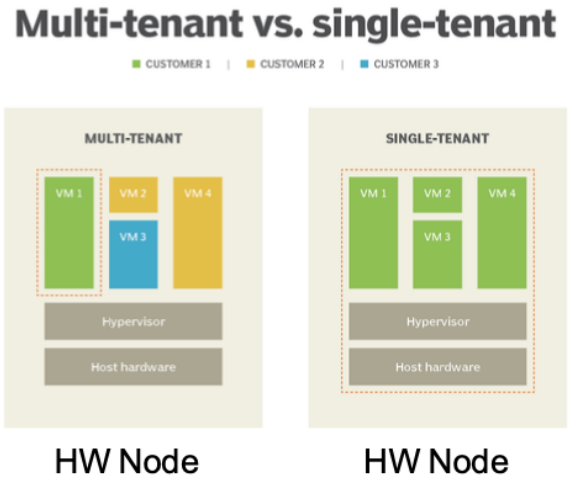


Figure 2.3: Resource pooling

- **Rapid elasticity** Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer,

the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.

- **Vertical and horizontal scaling**
  - **Vertical scaling**: adding more resources to a single node
  - **Horizontal scaling**: adding more nodes to a system, such as adding a new computer to a distributed software application
- **Measured service** Metering capability of service/resource abstractions in terms of storage, processing, bandwidth, active user accounts etc. Remember the utility computing and pay as you go model.

2.2 Cloud Service Models

- **Infrastructure as a Service (IaaS)**: provides virtualized computing resources over the internet. IaaS is one of the three main categories of cloud computing services, alongside Software as a Service (SaaS) and Platform as a Service (PaaS).
- **Platform as a Service (PaaS)**: provides a platform allowing customers to develop, run, and manage applications without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app.
- **Software as a Service (SaaS)**: is a software distribution model in which a third-party provider hosts applications and makes them available to customers over the internet.

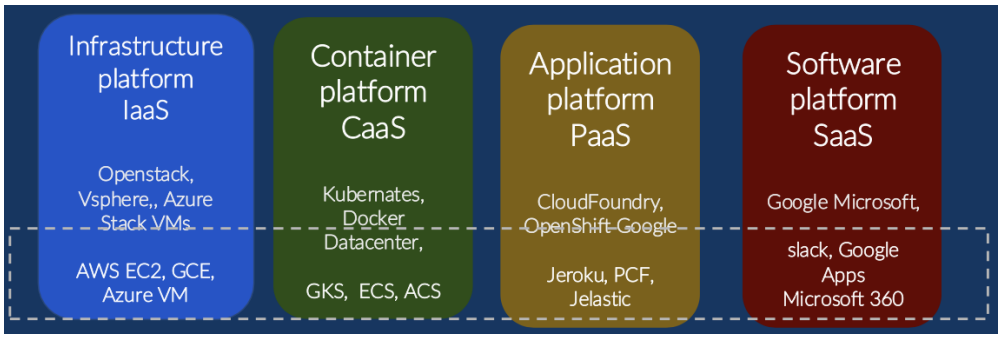


Figure 2.4: Cloud Service Models

2.3 Cloud Deployment Models

- **Public Cloud**: The cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. They exist on the premises of the cloud provider.
- **Commercial cloud**: The cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. They exist on the premises of the cloud provider.
- **Hybrid cloud**: The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds).
- **Community cloud**: A community cloud is one where the cloud has been organized to serve a common function or purpose.

2.4 Cloud Computing Infrastructure

- **Data Center**: A data center is a facility composed of networked computers and storage that businesses or other organizations use to organize, process, store and disseminate large amounts of data.
- **Virtualization**: Virtualization is the process of creating a virtual version of something, including virtual computer hardware platforms, storage devices, and computer network resources.
- **Hypervisor**: A hypervisor, also known as a virtual machine monitor, is a process that creates and runs virtual machines (VMs).
- **Containerization**: Containerization is a lightweight alternative to full machine virtualization that involves encapsulating an application in a container with its own operating environment.
- **Microservices**: Microservices are a software development technique—a variant of the service-oriented architecture (SOA) architectural style that structures an application as a collection of loosely coupled services.

It is based on the concept of virtualization, which allows for the creation of multiple virtual machines on a single physical machine. This allows for the efficient use of resources and the ability to scale up or down as needed. For this, Virtual Machines (VMs) are used, which are software-based representations of physical machines. These VMs can be created, modified, and deleted as needed, allowing for the efficient use of resources.

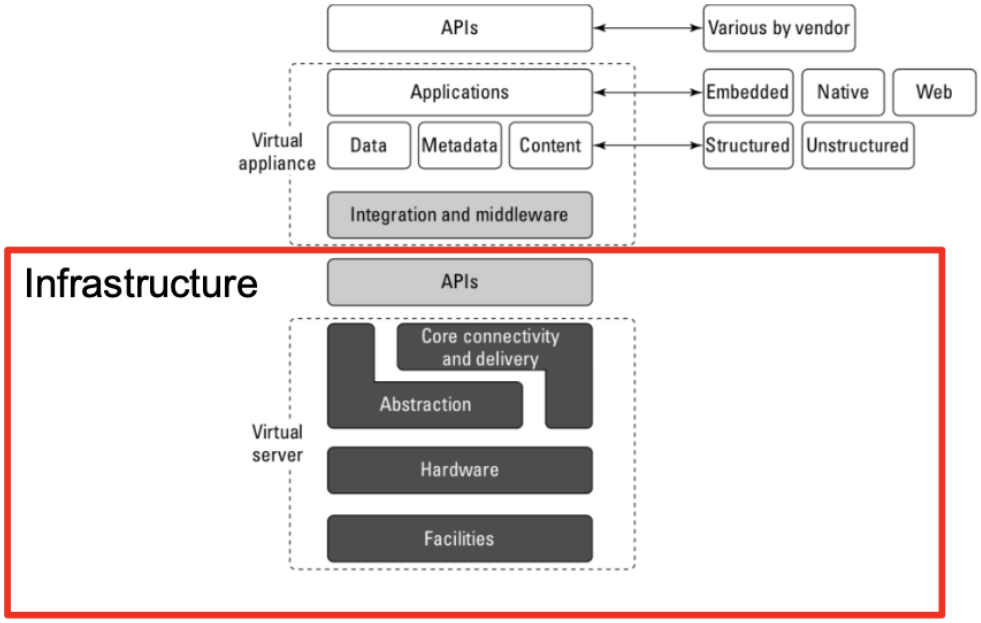


Figure 2.5: Cloud Computing Infrastructure

Applications such as a Web server or database server that can run on a virtual machine image are referred to as virtual appliances. Virtual appliances are software installed on virtual servers. They are self-contained and run on a virtual machine. They are pre-configured and ready to run applications.

2.5 Communication in Cloud Computing

Cloud computing arises from services available over the Internet communicating using the standard Internet protocol suite underpinned by the HTTP and HTTPS transfer protocols.

- **HTTP**: Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web.
- **HTTPS**: Hypertext Transfer Protocol Secure (HTTPS) is an extension of the Hypertext Transfer Protocol (HTTP). It is used for secure communication over a computer network and is widely used on the Internet.
- **REST**: Representational State Transfer (REST) is a software architectural style that defines a set of constraints to be used for creating Web services.
- **SOAP**: Simple Object Access Protocol (SOAP) is a messaging protocol that allows programs that run on disparate operating systems (such as Windows and Linux) to communicate using Hypertext Transfer Protocol (HTTP) and its Extensible Markup Language (XML).

Observation: ***REST***

REST is an architectural style that defines a set of constraints to be used for creating web services. RESTful web services allow the requesting systems to access and manipulate textual representations of web resources by using a uniform and predefined set of stateless operations.

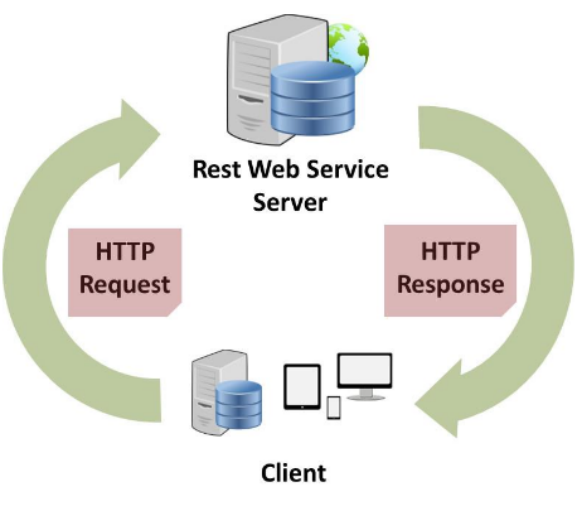


Figure 2.6: REST

Advantages of REST:

- **Scalability**: RESTful web services can be scaled to accommodate a large number of clients.
- **Performance**: RESTful web services are faster than SOAP web services because they are usually written in a lightweight language like JSON.
- **Simplicity**: RESTful web services are easier to understand and implement than SOAP web services.
- **Flexibility**: RESTful web services can be used with any programming language and can be easily integrated with other web services.

REST implementation:

1. Identify all the conceptual entities that we wish to expose as services.
2. Create a URL to each resource.
3. Categorize our resources according to whether clients can just receive a representation of the resource or whether clients can modify the resource.

4. All resources accessible via HTTP GET should be side-effect free, i.e., the resource should just return a representation of the resource (not modify it).
5. Put hyperlinks in the representation of the resource to allow clients to navigate to related resources.
6. Design to reveal data gradually. Don't reveal anything in a single response document. Provide hyperlinks to obtain mode details.
7. Specify the format of response datausing a schema (DTD, W3C Schema, RelaxNG, ...). For those services that require a POST or PUT to it, also provide a schema to specify the format of the response.
8. Describe how our services are to be invoked using either a WSDL document or simply an HTML document.

3 Computing basics

What is a computer?

A computer is a machine that can be programmed to automatically carry out sequences of arithmetic or logical operations (computation). Modern digital electronic computers can perform generic sets of operations known as programs. These programs enable computers to perform a wide range of tasks. The term computer system may refer to a nominally complete computer that includes the hardware, operating system, software, and peripheral equipment needed and used for full operation; A Computer cluster is a group of computers that are linked and function together.

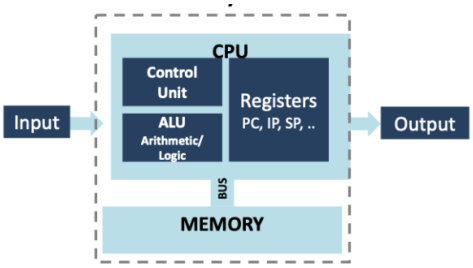


Figure 3.1: Serial Computer

A **serial computer** is a computer that processes data one bit at a time. This is in contrast to a parallel computer, which processes multiple bits at the same time. Serial computers are much slower than parallel computers, but they are also much simpler and cheaper to build.

Today, the most common form of computer is a digital computer, and computing machines have been an integral part of the business and engineering world since the late 20th century. Computers are used in a wide range of applications, including data processing, business management, and word processing. They are also used in scientific research, engineering, and medicine. Computers are used to control industrial processes and to simulate complex systems. Modern computers are capable of performing a wide range of tasks, from simple arithmetic to complex calculations.

A **parallel computer** is a computer that processes data multiple bits at the same time. This is in contrast to a serial computer, which processes data one bit at a time. Parallel computers are much faster than serial computers, but they are also much more complex and expensive to build.

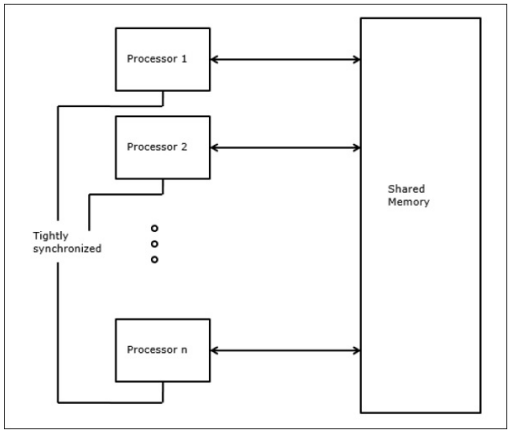


Figure 3.2: Parallel Computer

There are two main types of parallel computers: shared memory and distributed memory. In a shared memory system, all processors share the same memory. In a distributed memory system, each processor has its own memory. Parallel computers are used in a wide range of applications, including scientific research, engineering, and business.

- **Shared memory:** In a shared memory system, all processors share the same memory. This allows processors to communicate with each other by reading and writing to the same memory locations. Shared memory systems are typically used in applications that require high performance and low latency, such as scientific research and engineering.
  - Uniform Memory Access (UMA): each processor has uniform access to memory.

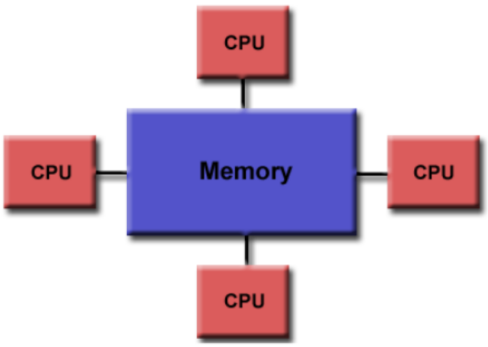


Figure 3.3: Uniform Memory Access (UMA)

  - Non-Uniform Memory Access (NUMA): time for memory access depends on location of data. Local access is faster than non-local access.

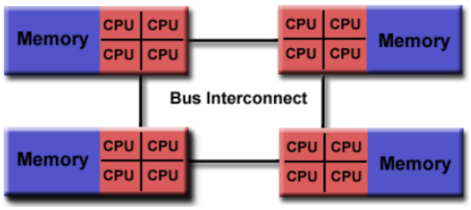


Figure 3.4: Non-Uniform Memory Access (NUMA)

- **Distributed memory:** In a distributed memory system, each processor has its own memory. This allows processors to communicate with each other by sending messages over a network. Distributed memory systems are typically used in applications that require scalability and fault tolerance, such as large-scale data processing and cloud computing.

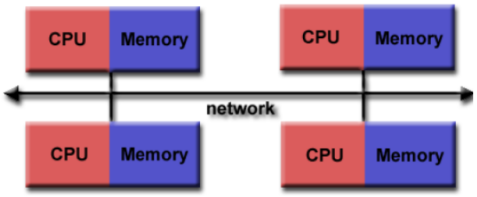


Figure 3.5: Distributed Memory

3.1 Cluster

A computer cluster is a group of computers that are linked and function together. The components of a cluster are usually connected to each other through fast local area networks, with each node running its own instance of an operating system. Clusters are used for a wide range of applications, including scientific research, engineering, and business. They are also used in cloud computing and high-performance computing.

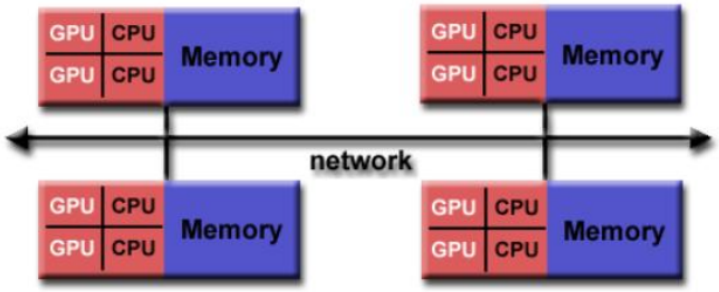


Figure 3.6: Computer Cluster

Compopnents of a cluster:

- **Nodes:** A node is a single computer in a cluster. Each node has its own processor, memory, and storage. Nodes are connected to each other through a network.
- **Network:** The network is the communication infrastructure that connects the nodes in a cluster. The network allows nodes to communicate with each other and share data.
- **Operating System:** Each node in a cluster runs its own instance of an operating system. The operating system manages the resources of the node and provides an interface for running applications.
- **Applications:** Applications are programs that run on the nodes in a cluster. Applications can be parallelized to take advantage of the resources of the cluster.

Figure 3.7: Components of a Cluster

What does a node contain?

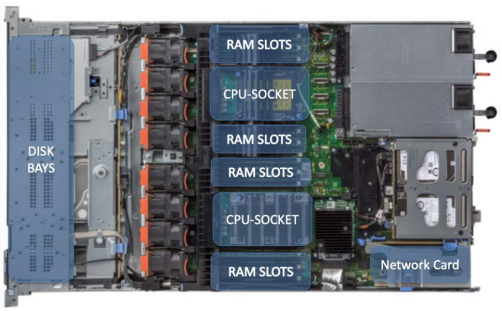


Figure 3.8: Node Components

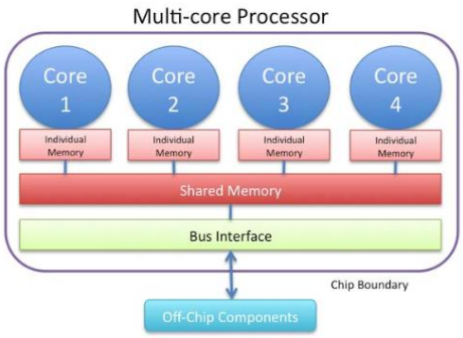


Figure 3.9: multicore CPU

CPU are multicore proccessors. Because of power, heat dissipation, ..., increasing tendency to actually lower clock frequency but pack more computing cores onto a chip. These cores will share some resources, i.e. memory, network, disk, etc. but are still capable of independent calculations.

Definition: *Core*

A core is the smallest unit of computing, having one or more (hardware/software) threads and is responsible for executing instructions.

Intel ®Hyper-Threading Technology uses processor resources more efficiently, enabling multiple threads to run on each core. The OS ”sees two cores and transparently try to execute two programs on two different ”cores”.

A bit of jargon:

- **Multiprocessor** = server with one or more than 1 CPU
- **Multicore** = CPU with more than 1 core
- **Processor** = CPU = socket = chip

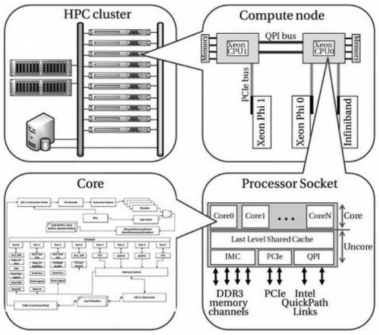


Figure 3.10: Computing Cluster

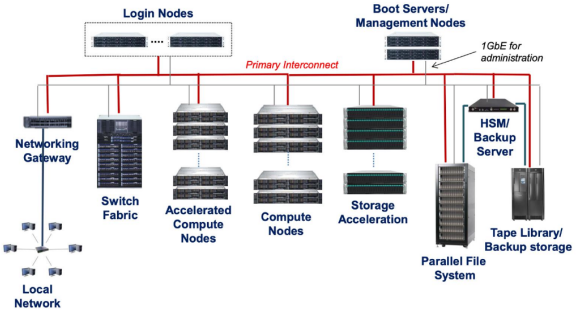


Figure 3.11: HPC cluster

The **scheduler** is a software component that manages the resources of a cluster. It is responsible for allocating resources to applications and ensuring that they run efficiently. The scheduler is typically part of the operating system of the cluster and is responsible for managing the resources of the cluster, such as CPU, memory, and storage. The scheduler uses algorithms to allocate resources to applications based on their requirements and priorities.

- It allocates exclusive or non-exclusive access to the resouces (the nodes) to users during a limited amout of time so that they can perform their work
- It abitrates contention for resources by managina a queue of pending work
- It permits to schedule jobs for users on the cluster resources

A user job is characterized by:

- the number of nodes
- the number of CPU cores
- the memory requested
- the walltime
- the launcher script, which will initiate your task

A **partition** is a group of compute nodes, with specific usage characteristics, like time limits and maximum number of nodes per job.

4 Virtualization

Virtualization uses software to create an *abstraction layer* over computer hardware that allows the hardware elements of a single computer - processors, memory, storage and more - to be divided into multiple virtual computers, commonly called virtual machines (VMs). Each VM runs its own operating system (OS) and behaves like an independent computer, even though it is running on just a portion of the actual underlying computer hardware.

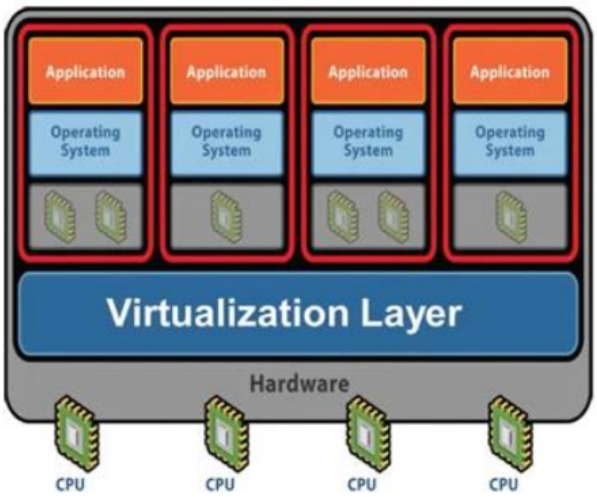


Figure 4.1: Virtualization

It is the ability to "simulate" a hardware platform, such as a server, storage device or network resource, in software. All the functionality is separated (abstracted) from the hardware and "simulated" as a "virtual instance" with the ability to operate just like the hardware solution. A single hardware platform can be used to support multiple virtual devices or machines, which are easy to spin up or down as needed.

Virtual Machine refers to a software simulation of a computer. It can run an OS and applications interacting with the virtualized abstracted resources, *not with the physical resources* of the actual host computer.

Hypervisor (or VM monitor) refers to a software tool installed on the physical host system to provide the this software layer of abstraction that decouples the OS from the physical bare-metal. It allows to split a computer in different separate environments, the VMs, distributing them the computer resources.

Components:

- **Host OS**: the OS running on the physical machine.
- **Hypervisor**: the software layer that abstracts the hardware and creates the VMs.
- **Guest OS**: the OS running on the VM.

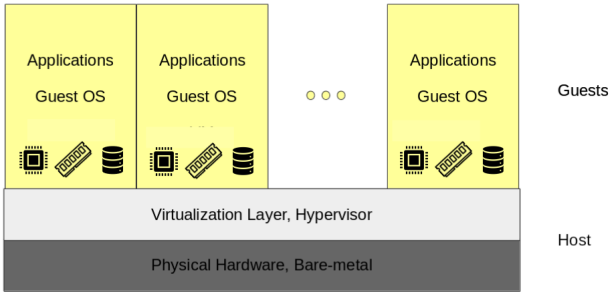


Figure 4.2: Virtualization components

4.1 Virtual Machines

A Virtual Machine is a virtual computing system. It has tighly **isolated** software with an OS and applications inside. Each VM in a host is **independent**.
In a single physical server can be put multiple VMs enabling the run of multiple OSes and Applications (*partition/multi-tenancy*).
Features:

- **Consolidation**: multiple VMs on a single physical server.
- **Isolation**: VMs are independent.
- **Encapsulation**: VMs are portable.
- **Hardware Independence**: VMs are not tied to the physical hardware.
- **Security**: VMs are isolated from each other.

4.2 Virtualization models

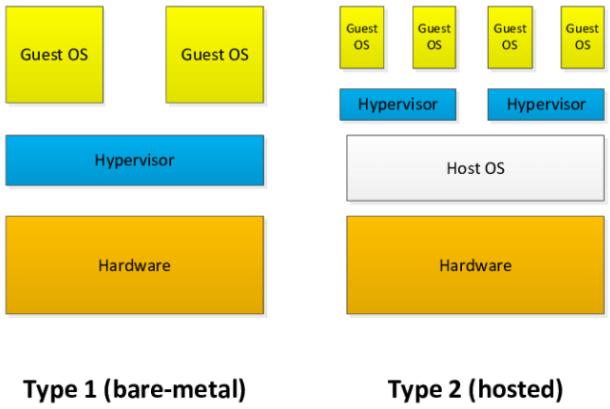


Figure 4.3: Virtualization models

1. **Type 1 Hypervisor**: it runs directly on the host's hardware to control the hardware and to manage guest operating systems. It is also called *bare metal hypervisor*. It is the most efficient because it has direct access to the hardware. Examples: VMware ESXi, Microsoft Hyper-V, Citrix XenServer, Oracle VM Server for SPARC.
2. **Type 2 Hypervisor**: it runs on a conventional operating system just as other computer programs do. It is also called *hosted hypervisor*. It is easier to set up and use, but it is less efficient because it must use the host OS to access the hardware. Examples: VMware Workstation, VMware Player, Oracle VirtualBox, Parallels Desktop for Mac.

Another distinction can be made w.r.t. virtualization:

- **Full virtualization**: the hypervisor provides complete hardware abstraction creating simulated hardware devices. The guest OS don't know (or care) about the presence of a hypervisor and issue commands to what it thinks is actual hardware.
- **Paravirtualization**: the guest OS is aware of the hypervisor and interacts with it
- **Hardware-assisted virtualization**: the hypervisor uses hardware capabilities to

Warning: *Hardware Protection Levels*

Since computers run more than on software process, this will bring some issues. Protection rings are one of the key solutions for sharing resources and hardware.

- **Ring 0**: the most privileged level (kernel mode).
- **Ring 1-2-3**: less privileged levels (user mode).

The hypervisor runs in Ring 0, the guest OS runs in Ring 1-2-3.

4.3 Virtualizing components

4.3.1 CPU

The hypervisor must manage the CPU resources. It must decide which VM gets CPU time and when. It must also manage the CPU instructions that are executed by the VMs.

Guest instructions are executed directly by the hardware as much as possible: virtual machine monitor does not interfere with every single instruction that is issued by the guest operating system, or its applications. The non-privileged instructions will operate at hardware speeds.

Privileged instructions: Whenever a privileged instruction gets accessed, then the processor causes a trap, and control is automatically switched to the most privileged level, that is the hypervisor. At this point, the hypervisor can determine whether the operation is to be an allowed or not. Illegal operations will cause actions on the VM (as KILL), legal operations the hypervisor should perform the necessary emulation so that the guest operating system is under the impression that it does have control over the hardware.

4.3.2 Memory

Full virtualization: the guest operating system continues to observe a contiguous linear physical address space that starts from physical address 0. Three types of addresses:

- **Virtual addresses**, these are the ones that are used by the applications in the guest
- **Physical addresses**, these are the ones that the guest thinks are the addresses of the physical resources
- **The machine addresses**, these are the actual machine addresses with the actual physical addresses on the underlying platform.

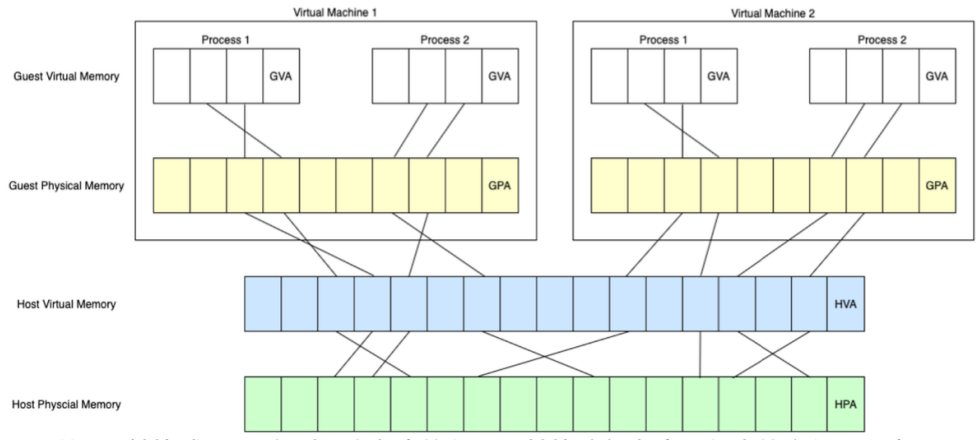


Figure 4.4: Memory virtualization

Every single memory access goes through two separate translation, the first one which will be done in software, and then the second one potentially can take advantage of hardware.
Moreover, **Shadow Page Table** estabilishes a shortcut to directly manage the mapping from GVA to HPA. It works the same way virtual addresses is mapped to physical addresses. VMM must maintain consistence between the page tables.

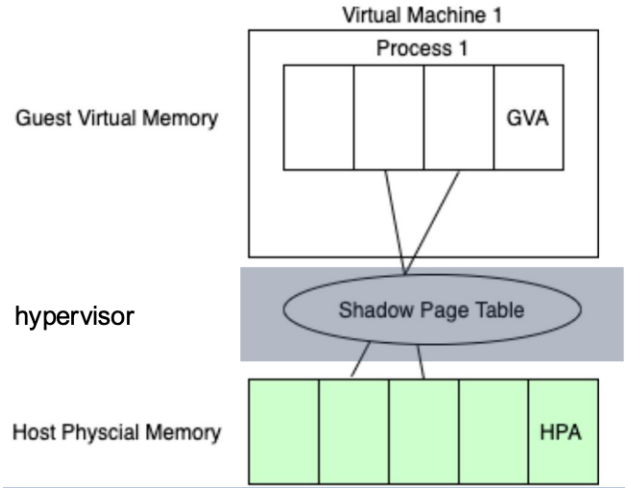


Figure 4.5: Shadow Page Table

Paravirtualization allows guest operating systems to be aware of their virtual environment and communicate directly with the hypervisor to manage memory. Instead of fully abstracting the underlying hardware, the guest OS uses an optimized interface to reduce the overhead of trapping into privileged operations. This approach decreases the complexity of mapping guest physical addresses to machine addresses by enabling the hypervisor and guest OS to collaborate on page table management and memory access, leading to more efficient performance overall.

4.3.3 Network

The hypervisor must manage the network resources. It must decide which VM gets network access and when. It must also manage the network packets that are sent and received by the VMs. Applications run on a virtual network as they where running on a physical network.

- **Flexibility**: VMs can be moved between physical servers without changing the network configuration.
- **Manageability**: VMs can be managed as a single entity.
- **Scalability**: VMs can be added or removed as needed.
- **Security**: VMs are isolated from each other.
- **Programmability**: VMs can be controlled by software.
- **Heterogeneity**: VMs can run different OSes and applications.

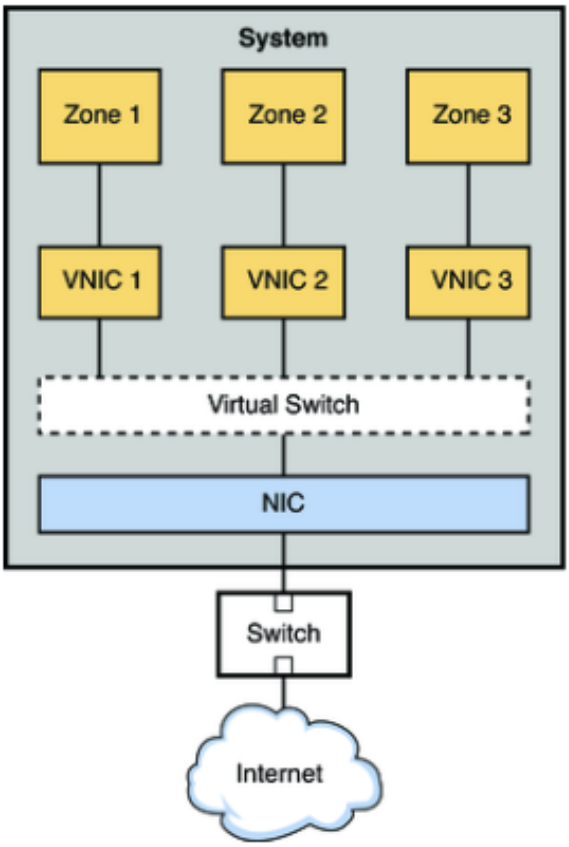


Figure 4.6: Network virtualization

4.3.4 Disk

The hypervisor must manage the disk resources. It must decide which VM gets disk access and when. It must also manage the disk blocks that are read and written by the VMs. Applications run on a virtual disk as they where running on a physical disk. Virtual Disks are where guest operating systems are installed, making them the equivalent of traditional hard disks. They are stored as files on the host system's physical disk.

- **Flexibility**: VMs can be moved between physical servers without changing the disk configuration.
- **Manageability**: VMs can be managed as a single entity.
- **Scalability**: VMs can be added or removed as needed.
- **Security**: VMs are isolated from each other.
- **Programmability**: VMs can be controlled by software.
- **Heterogeneity**: VMs can run different OSes and applications.

4.4 Emulation

Emulation is the process of simulating hardware using software. It is used when the guest OS is not aware of the hypervisor. The hypervisor must emulate the hardware that the guest OS expects to see.

Emulation brings higher overhead but has its perks too. It is highly inexpensive, easy to access, and helps us run the programs that have become obsolete in the available system. Anyone can access the emulation platforms remotely and is easier to use. It is an excellent ability to have for embedded/OS development, without affecting the underlying OS.

5 Benchmarking a Linux platform

Benchmarking is the process of measuring the performance of a system. It is a critical step in the process of designing and deploying a system. It is important for:

- **Maximize efficiency**: understanding how the server utilizes the CPU, memory, storage and network resources allows for fine-tuning and making better use of the hardwarre full potential.
- **Cost optimization**: understanding the performance of the system allows to make better decisions on the hardware to buy.
- **Troubleshooting issues**: benchmarking can help to identify bottlenecks and performance issues.
- **Security**: benchmarking can help to identify security issues.
- **Comparing systems**: benchmarking can help to compare different systems.

5.1 Components to test

There are a lot of tools available on the Internet that can perform system testing on Linux. We can use them to run different tests and benchmark various components, such as the Central Processing Unit (CPU), the Graphics Processing Unit (GPU), memory, database, and more.

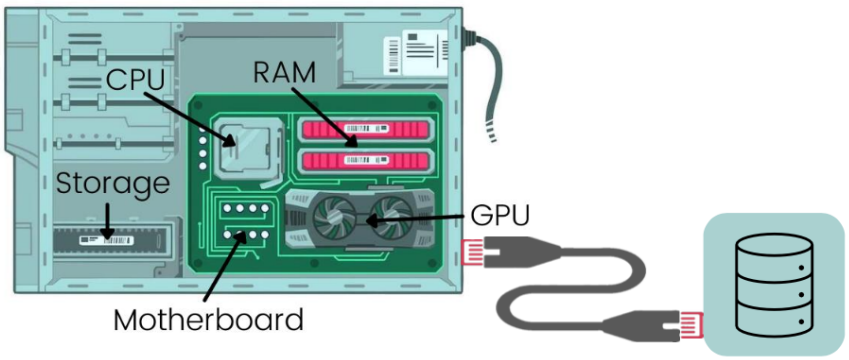


Figure 5.1: Components to test

- **CPU**: the CPU handles all computational tasks. Measuring its performances helps in understanding how efficiently it handles processes and threads, or whether there are bottlenecks in processing power.
  - stress-ng
  - sysbench
  - htop
  - mpstat
  - HPL
  - HPCG
  - SPEC

- **Memory (RAM)**: the memory is used to store data and instructions that are currently in use. Measuring its performances helps in understanding how efficiently it handles data and instructions, or whether there are bottlenecks in memory usage.
  - stress-ng
  - memtester
  - memtest86
  - valgrind
  - cachegrind
  - callgrind
- **Dist I/O and Storage**: the storage is used to store data and instructions that are not currently in use. Measuring its performances helps in understanding how efficiently it handles data and instructions, or whether there are bottlenecks in storage usage.
  - fio
  - ioping
  - dd
  - hdparm
  - bonnie++
  - iozone
- **Network**: the network is used to transfer data and instructions between different systems. Measuring its performances helps in understanding how efficiently it handles data and instructions, or whether there are bottlenecks in network usage.
  - iperf
  - netperf
  - nttcp
  - ping
  - traceroute
  - mtr

5.2 Benchmarking tools

5.2.1 HP Linpack (HPL)

This is the most famous HPC benchmark, used for the ”Top500” ranking. It is a software package that solves a random system of linear equations using Gaussian elimination (matrix is dense).
For a given problem size N, HPL performs floating point operations proportional to N*N*N operations while performing memory reads and write proportional to N*N. That means, if the problem size doubles, the number of floating-point operations increases by a factor of 8 while the number of memory operations only grows by a factor of 4. This property of HPL means that it performs very well on systems that have many flop functional units relative to data movement support. The result is that HPL tends to represent a very optimistic performance prediction that can only be matched by a small number of real scientific applications.

5.2.2 High Performance Conjugate Gradient (HPCG)

HPCG uses a simple implementation of the Conjugate Gradients and multigrid algorithms. It generates and uses sparse data structures that have a very low compute-to-data-movement ratio.

HPCG has a floating-point operations rate proportional to N and a memory access rate also proportional to N. This property means that HPCG performance is strongly influenced by memory bandwidth.

5.3 Tests

5.3.1 CPU

The CPU is responsible for executing all the processes, calculations, and tasks. A server's overall speed, responsiveness, and ability to handle concurrent workloads are highly dependent on the efficiency and power of its CPU.
In a virtualized environment, CPU testing becomes even more critical since physical CPU resources are shared between the virtual machines (VMs) running on the host. The virtual CPUs (vCPUs) allocated to each VM must be carefully monitored to ensure that the host and all guest VMs operate efficiently without resource contention.
To test it, we consider the Floating Point Operations Per Second (FLOPS):

$$FLOPS = \frac{Number\ of\ floating\ point\ operations}{Time}$$

Floating point operations per clock cycle per core is CPU dependent.
When testing CPU performance, several key metrics provide insight into how well the CPU handles the system's wordload.

- **CPU utilization**: the percentage of time the CPU is busy processing instructions.
- **CPU load**: the number of processes in the run queue.
- **CPU temperature**: the temperature of the CPU.
- **CPU frequency**: the frequency of the CPU.
- **CPU cache**: the cache of the CPU.
- **CPU Wait Time**: the time the CPU is waiting for I/O operations to complete.

Some tools to test the CPU are:

1. **Baseline tests**: Start by measuring CPU performance under typical workloads to establish a baseline. This allows you to understand how the CPU performs under normal conditions. Use as example `HPL` (see HPL Calculator).
2. **Stress tests**: Push the CPU to its limits to see how it performs under heavy workloads. Use as example `stress-ng` or `sysbench`. It can create stress on CPU, cache and memory.
3. **htop**: An interactive system monitor that provides real-time information on CPU utilization, load average, and the number of running processes. It visually represents per-core utilization. Simply run `htop` from the command line.
4. **mpstat**: A command-line utility that provides real-time CPU statistics. It displays information on CPU utilization, idle time, and wait time. Run `mpstat` from the command line.

Analysis on the results:

- **CPU utilization**: the percentage of time the CPU is busy processing instructions. A high CPU utilization indicates that the CPU is working hard to process instructions. A low CPU utilization indicates that the CPU is not working hard to process instructions.
- **CPU load**: the number of processes in the run queue. A high CPU load indicates that there are many processes waiting to be executed. A low CPU load indicates that there are few processes waiting to be executed.
- **High Context Switches**: A high number of context switches indicates that the CPU is switching

between processes frequently. This can lead to performance issues.

- **High Interrupts**: A high number of interrupts indicates that the CPU is handling a large number of hardware interrupts. This can lead to performance issues.
- **vCPU Overcommitment**: Overcommitting vCPUs can lead to performance issues. Make sure that the number of vCPUs allocated to a VM does not exceed the number of physical cores on the host.

5.3.2 Memory

Memory is used to store data and instructions that are currently in use. Measuring its performance helps in understanding how efficiently it handles data and instructions, or whether there are bottlenecks in memory usage.

Memory is one of the most vital components of a server, affecting how quickly applications can access and manipulate data. Inadequate memory resources, poor memory management, or issues with memory speed can lead to severe performance bottlenecks. For example, if the system runs out of physical memory, it may start using swap space (disk-based memory), which can significantly degrade performance.

Testing memory performance ensures that the system can handle workloads effectively without encountering delays due to insufficient or slow memory. This is particularly important in environments running databases, virtual machines, or in- memory data stores (like Redis).

When testing memory performance, several key metrics provide insight into how well the memory handles the system's workload.

- **Memory utilization**: the percentage of memory that is being used.
- **Memory load**: the number of processes that are using memory.
- **Memory latency**: the time it takes to access memory.
- **Memory bandwidth**: the rate at which data can be read from or written to memory.
- **Memory errors**: the number of memory errors that occur.
- **Cache Hit/Miss Ratio**: the ratio of cache hits to cache misses.

Some tools to test the memory are:

1. **Baseline tests**: Start by measuring memory performance under typical workloads to establish a baseline. This allows you to understand how the memory performs under normal conditions. Use as example `memtester` or `memtest86`.
2. **Stress tests**: Push the memory to its limits to see how it performs under heavy workloads. Use as example `stress-ng` or `memtester`.
3. **valgrind**: A memory debugging tool that can be used to profile memory usage and detect memory leaks. Run `valgrind` from the command line.
4. **cachegrind**: A cache profiling tool that can be used to profile cache usage. Run `cachegrind` from the command line.
5. **callgrind**: A call graph profiling tool that can be used to profile function calls. Run `callgrind` from the command line.

Analysis on the results:

- **Memory utilization**: the percentage of memory that is being used. A high memory utilization indicates that the memory is being used heavily. A low memory utilization indicates that the memory is not being used heavily.
- **Memory load**: the number of processes that are using memory. A high memory load indicates that there are many processes using memory. A low memory load indicates that there are few

processes using memory.

- **Memory latency**: the time it takes to access memory. A low memory latency indicates that memory access is fast. A high memory latency indicates that memory access is slow.
- **Memory bandwidth**: the rate at which data can be read from or written to memory. A high memory bandwidth indicates that memory access is fast. A low memory bandwidth indicates that memory access is slow.
- **Memory errors**: the number of memory errors that occur. A high number of memory errors indicates that there are issues with memory.
- **Cache Hit/Miss Ratio**: the ratio of cache hits to cache misses. A high cache hit/miss ratio indicates that the cache is being used effectively. A low cache hit/miss ratio indicates that the cache is not being used effectively.

5.3.3 Disk I/O and Storage

Disk I/O (Input/Output) is a critical factor in server performance, especially for databases, web servers, and applications that rely on frequent disk access. Slow disk performance can lead to bottlenecks where applications are waiting on data retrieval or writing operations, causing delays and system slowdowns. In addition, storage subsystems like RAID arrays and SSDs have different performance characteristics that must be optimized for the server's workload.
Cloud and Virtualized environments also introduce additional complexity since multiple virtual machines may share the same physical storage, increasing the potential for resource contention.
When testing disk I/O and storage performance, several key metrics provide insight into how well the storage subsystem handles the system's workload.

- **Read/Write Throughput**: the rate at which data can be read from or written to disk.
- **Read/Write Latency**: the time it takes to read from or write to disk.
- **IOPS (Input/Output Operations Per Second)**: the number of read/write operations that can be performed per second.
- **Disk Queue Length**: the number of read/write requests that are waiting to be processed.
- **Disk Errors**: the number of disk errors that occur.
- **SSD vs HDD**: the performance difference between SSDs and HDDs.

Some tools to test the disk I/O and storage are:

1. **Baseline tests**: Start by measuring disk I/O and storage performance under typical workloads to establish a baseline. This allows you to understand how the storage subsystem performs under normal conditions. Use as example `fio` or `dd`.
2. **Stress tests**: Push the disk I/O and storage to its limits to see how it performs under heavy workloads. Use as example `fio` or `dd`.
3. **ioping**: A disk I/O latency tool that can be used to measure disk I/O latency. Run `ioping` from the command line.
4. **hdparm**: A disk performance tool that can be used to measure disk performance. Run `hdparm` from the command line.
5. **bonnie++**: A disk benchmarking tool that can be used to measure disk performance. Run `bonnie++` from the command line.
6. **iozone**: A disk benchmarking tool that can be used to measure disk performance. Run `iozone` from the command line.

Analysis on the results:

- **Read/Write Throughput**: the rate at which data can be read from or written to disk. A high

read/write throughput indicates that the disk I/O and storage subsystem is performing well. A low read/write throughput indicates that the disk I/O and storage subsystem is not performing well.

- **Read/Write Latency**: the time it takes to read from or write to disk. A low read/write latency indicates that disk I/O and storage access is fast. A high read/write latency indicates that disk I/O and storage access is slow.
- **IOPS (Input/Output Operations Per Second)**: the number of read/write operations that can be performed per second. A high IOPS indicates that the disk I/O and storage subsystem is performing well. A low IOPS indicates that the disk I/O and storage subsystem is not performing well.
- **Disk Queue Length**: the number of read/write requests that are waiting to be processed. A high disk queue length indicates that there are many read/write requests waiting to be processed. A low disk queue length indicates that there are few read/write requests waiting to be processed.
- **Disk Errors**: the number of disk errors that occur. A high number of disk errors indicates that there are issues with the disk I/O and storage subsystem.
- **SSD vs HDD**: the performance difference between SSDs and HDDs. SSDs are faster than HDDs and have lower latency.

6 Containers

Virtual Environments

Pros	Cons
• Reproducible research • Explicit dependencies • Improved engineering collaboration	• Difficulty setting up your environment • Not isolation • Does not always work across different OS

Virtual Machines

Pros	Cons
• Full autonomy • Very secure • Isolation • Lower costs • used by all Cloud providers	• Uses hardware in local machine • Not very portable since size of VMs are large • There is an overhead associated with virtual machines

Warning: *Dependency Hell*

- **Dependency Hell** is a colloquial term for the frustration of some software users who have installed software packages which have dependencies on specific versions of other software packages.
- It is a problem with software dependencies that occurs when a software package is designed to work with particular versions of other software packages.
- If a user wants to install a software package that has dependencies on specific versions of other software packages, it can be difficult to resolve these dependencies.

The solution for all these problems is using **Containers**.

Definition: *Container*

Informally, a **container** is a standard unit of software that packages up code and all its dependencies in processes isolated from resources, so the application runs quickly and reliably from one computing environment to another. Containers creates an isolated environment at application level and not at server level.

They come with different advantages:

- Extremely portable and lightweight
- Fully packaged software with all dependencies included
- Can be used for development, training and and deployment
- Development teams can easily share containers

A **container image** is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings. Available for both Linux and Windows based apps, containerized software will always run the same, regardless of the environment.

Container images become containers at runtime and in the case of Docker containers - images become containers when they run on Docker Engine. Available for both Linux and Windows-based applications, containerized software will always run the same, regardless of the infrastructure.

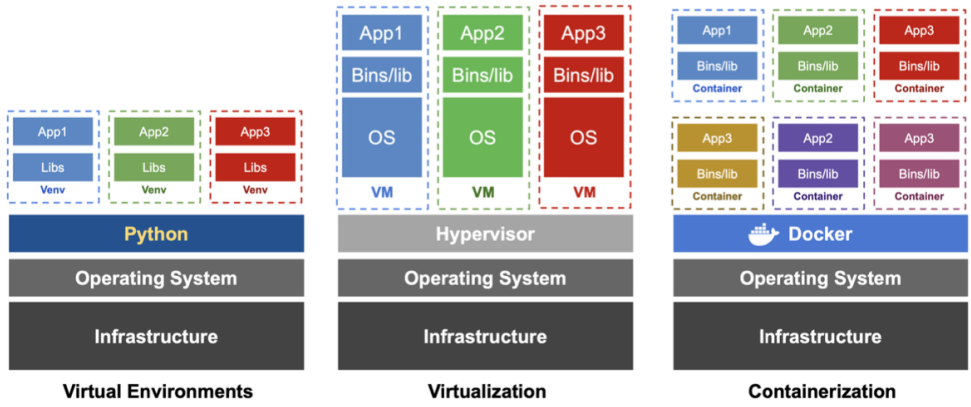


Figure 6.1: Comparison between Venvs, VMs and Containers

6.1 Properties

- **Isolation**: Containers virtualize CPU, memory, storage, and network resources at the OS-level, providing developers with a sandboxed view of the OS logically isolated from other applications. Developers, using containers, are able to create predictable environments isolated from other applications.
- **Productivity**: enhancement Containers can include software dependencies needed by the application (specific versions of programming language runtimes, software libraries) guaranteed to be consistent no matter where the application is deployed. All this translates to productivity: developers and IT operations teams spend less time debugging and diagnosing differences in environments, and more time shipping new functionality for users.
- **Deployment simplicity**: Containers can be deployed on any machine that has a container runtime installed. This means that developers can build and test containers on their local machine and then deploy them to any environment that supports containers. This makes it easy to move applications between environments, such as from a developer's laptop to a test environment, or from a test environment to production.
- **Portability**: Containers are portable because they include all of the dependencies needed to run the application. This means that containers can run on any machine that has a container runtime installed, regardless of the underlying infrastructure. This makes it easy to move applications between environments, such as from a developer's laptop to a test environment, or from a test environment to production.

- **Easy Versioning**: Containers make it easy to version applications. Developers can create a new version of an application by creating a new container image with the new version of the application. This makes it easy to roll back to a previous version of an application if there are any issues with the new version.
- **Scalability**: Containers are lightweight and can be started and stopped quickly. This makes it easy to scale applications up and down based on demand. For example, if an application is experiencing high traffic, additional containers can be started to handle the load. Once the traffic decreases, the additional containers can be stopped.
- **Operational efficiency and reliability**: Containers make it easy to automate the deployment and management of applications. This makes it easy to deploy applications quickly and reliably. Containers can also be used to automate the scaling of applications based on demand. This makes it easy to ensure that applications are always available and responsive.
- **Security**: Containers provide a level of isolation between applications that can help improve security. Containers can be used to run applications in a sandboxed environment that is isolated from other applications. This can help prevent applications from interfering with each other and can help prevent security vulnerabilities from being exploited.

The differences with the VMs are:

- Containers are more lightweight than VMs
- Containers are faster to start and stop than VMs
- Containers use less memory than VMs
- Containers are more portable than VMs
- Containers are easier to manage than VMs

You can find a practical example of these differences in my GitHub repository: Cluster Analysis.

6.2 Docker

Docker is a containerization platform that packages your application and all its dependencies together in the form of a docker container to ensure that your application works seamlessly in any environment.

Definition: ***Container Engine***

A **container engine** is a piece of software that accepts user requests, including command line options, pulls images, and from the end user's perspective runs the container. It is reponsible for:

- Building images
- Running containers
- Stopping containers
- Removing containers
- Managing networks
- Managing volumes

Docker Container is a standardized unit which can be created on the fly to deploy a particular application or environment. It could be an Ubuntu container, CentOs container, etc. to full-fill the requirement from an operating system point of view.

Containers create an isolated environment at application level: each container is a process running including all its children;

Containers underlying (main) technologies:

- Linux NAMESPACES: Isolates system resources
- Linux CGroups: Limits resources

User space refers to all the code in an operating system that lives outside of the kernel.

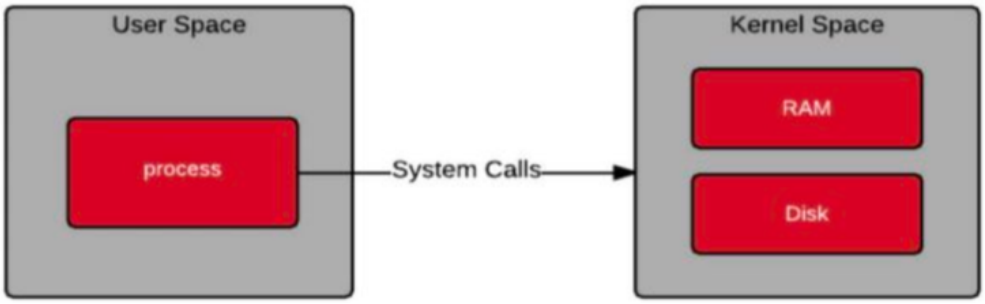


Figure 6.2: process Virtualization

6.2.1 Linux Namespaces

Linux namespaces provide a mechanism for isolating system resources, enabling processes within a namespace to have their own view of the system, such as process IDs, Namespaces in Linux provide a way to isolate and virtualize system resources, thus enhancing security by preventing processes in one namespace from directly interacting with processes in another namespace. Namespaces increase security by providing a level of isolation that prevents unintended interactions between processes. This isolation is particularly valuable in containerization and virtualization scenarios, where multiple applications or services share the same host system but must be kept separate for security reasons.

- A **user namespace** has its own set of user IDs and group IDs for assignment to processes. In particular, this means that a process can have root privilege within its user namespace without having it in other user namespaces.
- A **process ID (PID) namespace** assigns a set of PIDs to processes that are independent from the set of PIDs in other namespaces. The first process created in a new namespace has PID 1 and child processes are assigned subsequent PIDs. If a child process is created with its own PID namespace, it has PID 1 in that namespace as well as its PID in the parent process' namespace.
- A **network namespace** has an independent network stack: its own private routing table, set of IP addresses, socket listing, connection tracking table, firewall, and other network-related resources.
- A **mount namespace** has an independent list of mount points seen by the processes in the namespace. This means that you can mount and unmount filesystems in a mount namespace without affecting the host filesystem.

Within the parent namespace, there are four processes, named PID1 through PID4. These are normal processes which can all see each other and share resources.

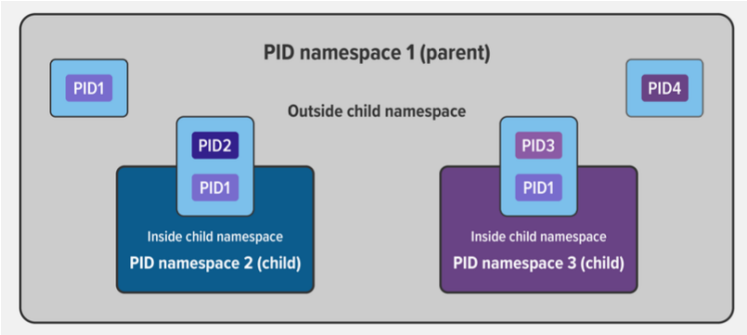


Figure 6.3: PIDs

To test namespaces, you can use the following commands:

```
# Show all processes
sudo ps -ef

# Show all namespaces
sudo lsns

# Create a new PID namespace
unshare --fork --pid --mount-proc bash

# Create a new network namespace
ip netns add test
ip netns exec test bash
```

Namespaces are one of the technologies that containers are built on, used to enforce segregation of resources. We've shown how to create namespaces manually, but container runtimes like Docker, rkt, and podman make things easier by creating namespaces on your behalf.

6.2.2 Linux CGroups

A control group (cgroup) is a Linux kernel feature that limits, accounts for, and isolates the resource usage (CPU, memory, disk I/O, network, and so on) of a collection of processes.
Main features are:

- **Resource limiting**: Set limits on the amount of resources a group of processes can use.
- **Prioritization**: Give certain groups of processes higher or lower priority.
- **Accounting**: Keep track of the resources used by a group of processes.
- **Control**: Freeze, resume, or kill a group of processes.

Kernel uses cgroups to control how much of a given key resource (CPU, memory, network, and disk I/O) can be accessed or used by a process or set of processes.

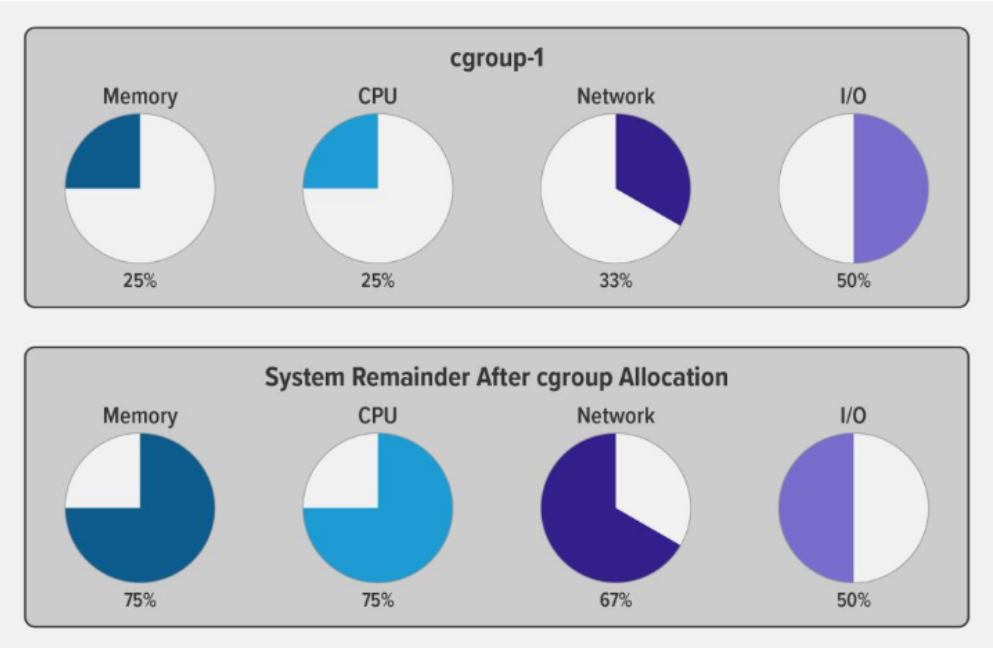


Figure 6.4: CGroups

You can access cgroups with various tools:

```
# Show all cgroups
sudo lscgroup

# Show cgroup hierarchy
systemd-cgtop

# Create a new cgroup
sudo cgcreate -g memory:mygroup

# Set memory limit
sudo cgset -r memory.limit_in_bytes=1G mygroup

# Add a process to a cgroup
sudo cgclassify -g memory:mygroup $PID

# Control groups can be implemented directly by container engines
docker run -d --name mycontainer --memory 1g myimage
```

6.2.3 Docker File

To build a Docker image, you need to create a Dockerfile. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Using docker build users can create an automated build that executes several command-line instructions in succession.

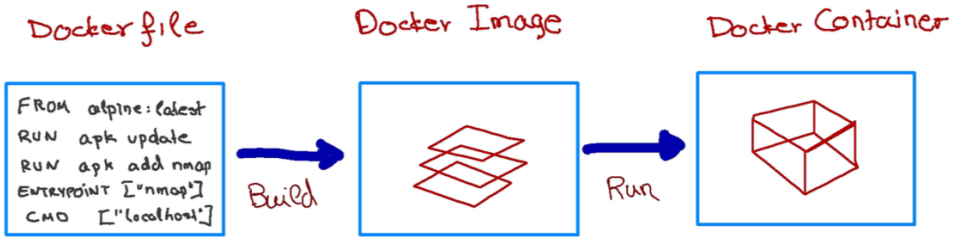


Figure 6.5: Dockerfile

The Dockerfile has the following structure:

- **FROM**: This instruction in the Dockerfile tells the daemon, which base image to use while creating our new Docker image. In the example here, we are using a very minimal OS image called alpine (just 5 MB of size). You can also replace it with Ubuntu, Fedora, Debian or any other OS image.
- **RUN**: This command instructs the Docker daemon to run the given commands as it is while creating the image. A Dockerfile can have multiple RUN commands, each of these RUN commands create a new layer in the image.
- **ENTRYPOINT**: The ENTRYPOINT instruction is used when you would like your container to run the same executable every time. Usually, ENTRYPOINT is used in scenarios where you want the container to behave exclusively as if it were the executable it's wrapping.
- **CMD**: The CMD sets default commands and/or parameters when a docker container runs. CMD can be overwritten from the command line via the docker run command.

Example: *Dockerfile*

```
1  FROM ubuntu:18.04
2  RUN apt update
3  RUN apt upgrade
4  ENTRYPOINT ["echo", "Hello, World!"]
```

You can create multiple containers from the same image, and each container will have its own isolated environment. You can also create multiple images from the same Dockerfile, and each image will have its own set of layers.

Docker images are based on layers. Each layer is a set of read-only files that is created when the image is built. When you run a container from an image, Docker creates a read-write layer on top of the image layers. This read-write layer is where the container writes data.

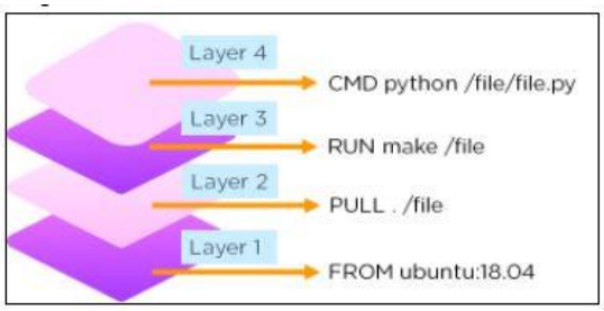


Figure 6.6: Docker Layers

7 Data Cloud and Cloud Security

7.1 Data Cloud

Cloud Storage is a service model in which data is maintained, managed and backed up remotely and made available to users over a network (typically the Internet). Users generally pay for their cloud data storage on a per-consumption, monthly rate. Although the per-gigabyte cost has been radically driven down, cloud storage providers have added operating expenses that can make the technology more expensive than users bargained for. Cloud security continues to be a concern among users. Providers have tried to address those fears by building security capabilities, such as encryption and authentication, into their services.
Traditional storages are:

Block Storage:
- Volumes
- Blocks (read/write)
- Fibre Channel or iSCSI protocols
- Local
- Low latency, high IOPs, low size (<1PB)
- Complex to expand and expensive

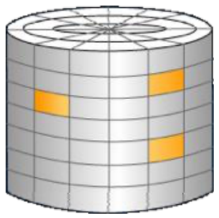


Figure 7.1: Block Storage

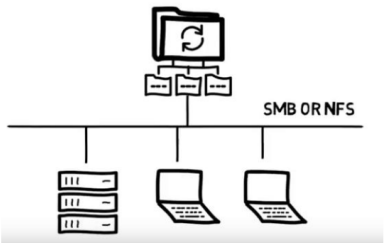


Figure 7.2: File Storage

File Storage:
- Files
- NFS, SMB, FTP, etc.
- Local
- Low latency, high IOPs, low size (<1PB)
- Complex to expand and expensive

7.1.1 Distributed File System

Definition: *Distributed File System*

A **Distributed File System** is a shared file system that allows many clients to access files and directories stored on a central server. It is a client/server-based application that allows clients to access and process data stored on the server as if it were on their own computer.
- **Client**: The client is the end-user device that accesses the files stored on the server.
- **Server**: The server is the computer that stores the files and directories that the clients access.

Properties:

- **Transparency**: A client cannot tell where a file is located. A file can transparently move to another server. Multiple copies of a file may exist. Multiple clients access the same file.
- **Flexibility**: Servers may be added or replaced and there is support for multiple file system types.
- **Dependability**: Conflicts with replication and concurrency control. Users may have different access rights on clients sharing files and network transmission. Server crashes may cause data loss.
- **Performance**: Requests may be distributed across servers and multiple servers allow higher storace capacity.
- **Caching**: Reduce network traffic by retaining recently accessed disk blocks in a cache, so that repeated accesses to the same information can be handled locally. If required data is not already cached, a copy of data is brought from the server to the user.

Configuration and implementation may vary, servers may run on dedicated machines or servers and clients can be on the same machines.

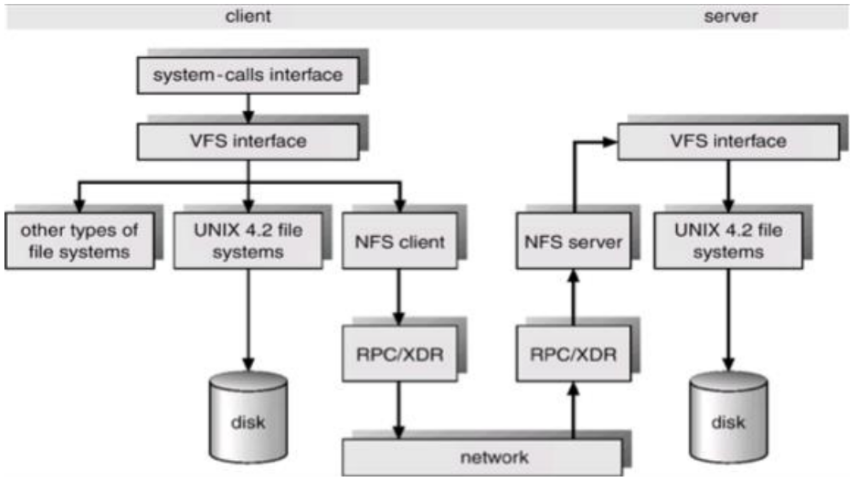


Figure 7.3: Distributed File System

Warning: *Limitations of traditional storage*

Traditional storage systems are limited in terms of scalability, performance, and cost. They are not designed to handle the massive amounts of data that are generated by modern applications. They are also not designed to handle the high levels of concurrency that are required by modern applications. In addition, traditional storage systems are expensive to purchase and maintain. They require a significant amount of hardware and software to be installed and configured, and they require a dedicated team of IT professionals to manage them.

Example: *Google File System*

The Google File System (GFS) is a distributed file system that was developed by Google to handle the massive amounts of data that are generated by its search engine. GFS is designed to be highly scalable, highly available, and highly reliable. It is designed to handle the high levels of concurrency that are required by modern applications. GFS is also designed to be cost-effective, with a low total cost of ownership. GFS is based on a master/slave architecture, with a single master server that manages the metadata for the file system and multiple slave servers that store the actual data. GFS uses a distributed lock service to manage access to

the file system, and it uses a distributed file system protocol to communicate between the master and slave servers. GFS is designed to be fault-tolerant, with multiple copies of each file stored on different servers to ensure that data is not lost in the event of a server failure. GFS is also designed to be highly available, with multiple master servers that can take over in the event of a master server failure. GFS is designed to be highly reliable, with built-in mechanisms for detecting and recovering from data corruption and other errors.

- **Master Server**: Single master server that manages the metadata for the file system.
- **Chunk Servers**: Multiple slave servers that store the actual data.

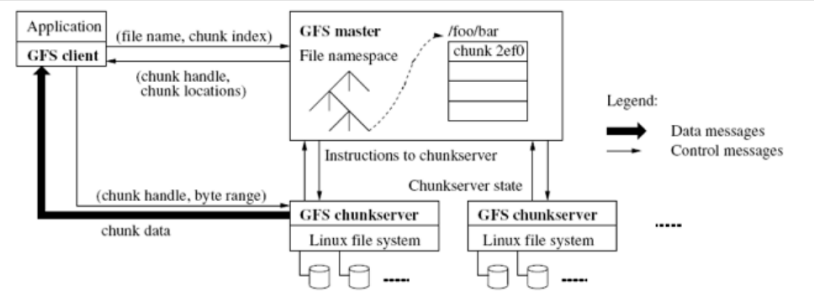


Figure 7.4: Google File System Architecture

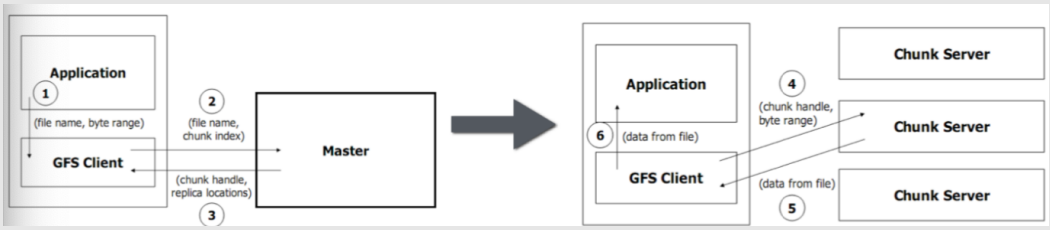


Figure 7.5: Read Files

An **object** is a logical unit of storage that is stored on a server. An object has a unique identifier, a set of attributes, and a set of methods that can be used to access and manipulate the object. Objects are typically stored in a distributed file system, where they are replicated across multiple servers to ensure high availability and durability. Objects are typically accessed using a RESTful API, which allows clients to perform CRUD operations on the objects using standard HTTP methods.

Metadata, instead, is data that describes the attributes of an object. Metadata is typically stored in a separate database or file system from the object itself, and is used to store information such as the object's name, size, type, and location. Metadata is typically accessed using a separate API from the object itself, which allows clients to query and update the metadata without having to access the object itself.

Observation: ***FITS files***

The Flexible Image Transport System (FITS) is an open standard defining a digital file format useful for storage, transmission and processing of data: formatted as multi-dimensional arrays (images) or tables. FITS is the most commonly used digital file format in astronomy. FITS is much more than just another image format (such as JPG or GIF) and is primarily designed to store scientific data sets consisting of multi-dimensional arrays (2D tables, 3D

data cubes, etc.) and 1D tables consisting of columns and rows. FITS is also often used to store non-image data, such as spectra, photon lists, data cubes, or structured data such as object catalogs.

Request for storage are made with HTTP using RESTful APIs. The request is made to the server, which is responsible for storing the data. The server then stores the data in a distributed file system, which replicates the data across multiple servers to ensure high availability and durability. The server then returns a response to the client, which indicates whether the request was successful or not. Primary components of a request are:

1. HTTP Verb: GET, POST, PUT, DELETE
2. Authentication Information
3. SURL: Storage URL
4. Metadata (optional)

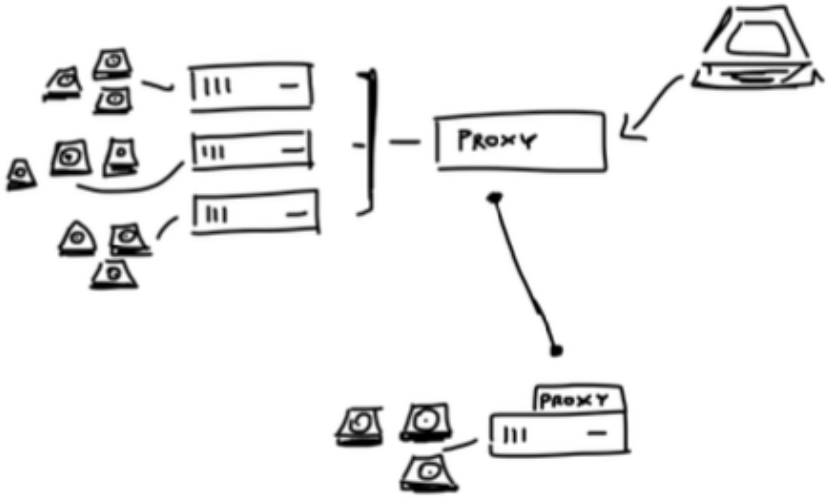


Figure 7.6: Request for Storage

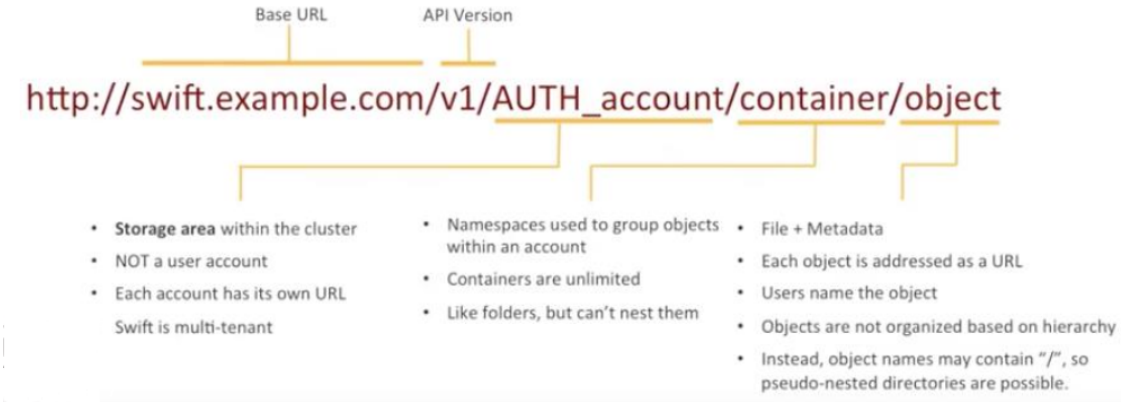


Figure 7.7: manage Object with HTTP

7.1.2 Cloud Storage capabilities

Definition: *Cloud Storage*

Cloud Storage is a service model in which data is maintained, managed and backed up remotely and made available to users over a network (typically the Internet). Users generally pay for their cloud data storage on a per-consumption, monthly rate. Although the per-gigabyte cost has been radically driven down, cloud storage providers have added operating expenses that can make the technology more expensive than users bargained for. Cloud security continues to be a concern among users. Providers have tried to address those fears by building security capabilities, such as encryption and authentication, into their services.

Its capabilities are:

- **Cloud File Storage**: Store and manage files in the cloud. Files are stored in a distributed file system, which replicates the data across multiple servers to ensure high availability and durability. Files are typically accessed using a RESTful API, which allows clients to perform CRUD operations on the files using standard HTTP methods.
- **User management**: Manage users and their access to files. Users are typically authenticated using a username and password, and their access to files is controlled using access control lists (ACLs). Users can be assigned different roles and permissions, which determine what actions they can perform on the files.
- **Easy to use GUI**: A graphical user interface (GUI) that allows users to easily upload, download, and manage files in the cloud. The GUI typically provides a file browser that allows users to navigate the file system, and a set of buttons that allow users to perform common file operations.
- **Security**: Security features such as encryption, authentication, and access control. Files are typically encrypted at rest and in transit, to protect them from unauthorized access. Users are authenticated using a username and password, and their access to files is controlled using access control lists (ACLs).
- **Scalability**: Scalability features such as automatic scaling and load balancing. The cloud storage system is designed to automatically scale to handle large amounts of data and high levels of concurrency. Load balancing ensures that requests are distributed evenly across the servers, to prevent any one server from becoming overloaded.
- **Multiple access protocols**: Support for multiple access protocols, such as RESTful APIs, NFS, and SMB. Users can access files using a variety of protocols, depending on their needs and preferences. RESTful APIs are typically used for programmatic access, while NFS and SMB are typically used for file sharing and collaboration.

Its architecture is based on a distributed file system, which replicates the data across multiple servers to ensure high availability and durability. The distributed file system is typically accessed using a RESTful API, which allows clients to perform CRUD operations on the files using standard HTTP methods. The distributed file system is typically implemented using a master/slave architecture, with a single master server that manages the metadata for the file system and multiple slave servers that store the actual data.

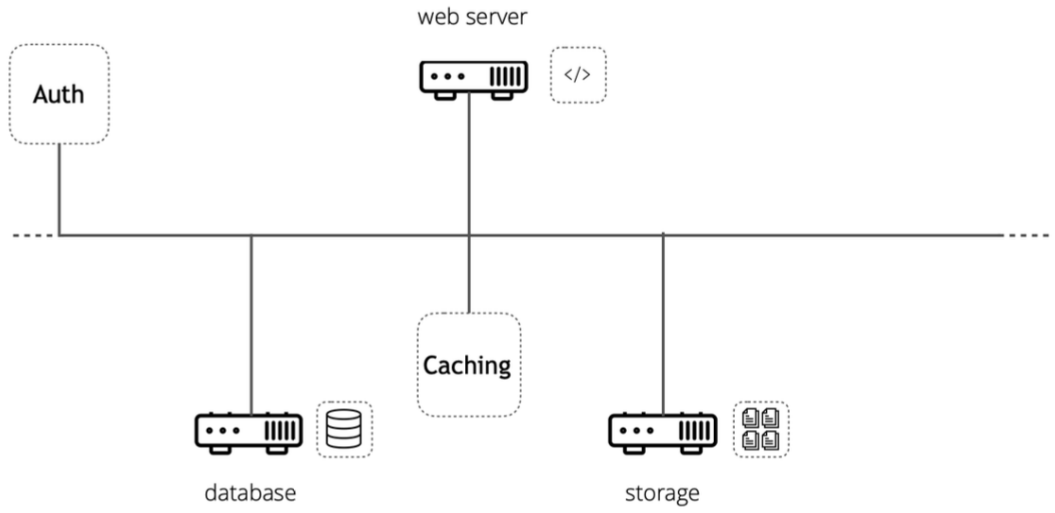


Figure 7.8: Cloud Storage Architecture

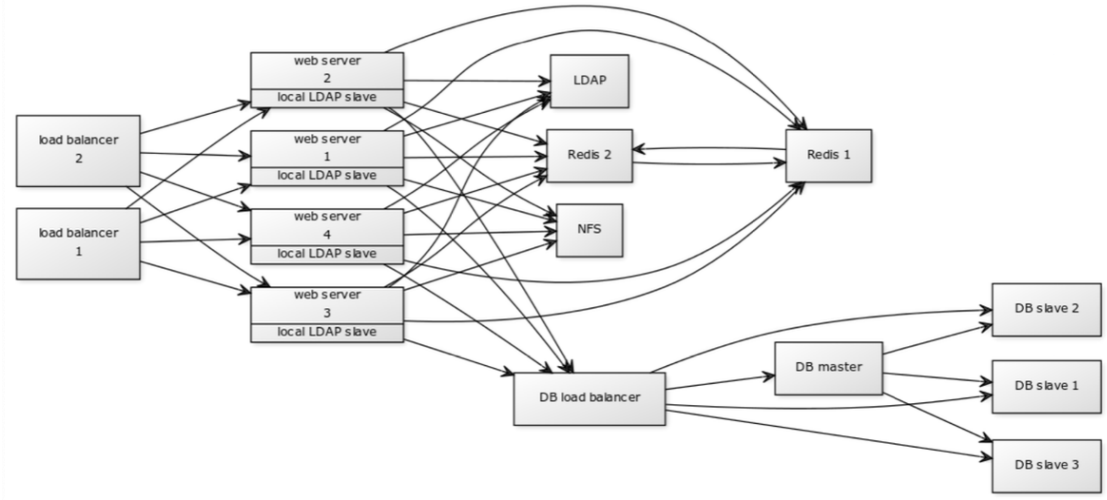


Figure 7.9: Production Deployment

To **monitor** the system several tools can be used:

- **Docker stats**: Command line tool that provides real-time resource usage statistics for Docker containers.
- **Portainer**: Management tool for Docker environments, easy to setup.
- **Grafana**: Analytics and visualization platform that allows users to query, visualize and monitor data from multiple sources in real-time. Requires configuration.

7.2 Cloud Security

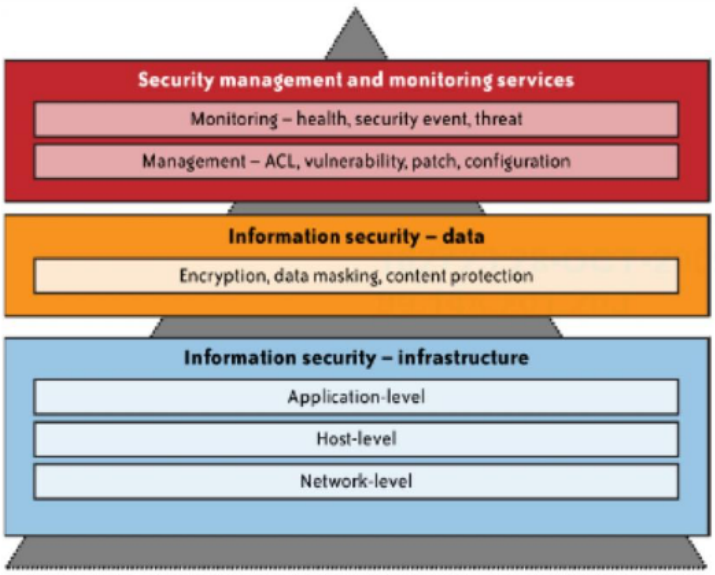


Figure 7.10: Cloud Security

Different types of cloud computing service models provide different levels of security. There are three levels of cloud security:

- **Network Level**: Security at the network level is the most basic level of security. It involves securing the network infrastructure that connects the cloud servers to the Internet. This includes firewalls, intrusion detection systems, and other network security devices. Network security is important because it protects the cloud servers from attacks that originate from the Internet.
- **Host Level**: Security at the host level is the next level of security. It involves securing the individual servers that make up the cloud infrastructure. This includes securing the operating system, the applications that run on the server, and the data that is stored on the server. Host security is important because it protects the cloud servers from attacks that originate from within the cloud infrastructure.
- **Application Level**: Security at the application level is the highest level of security. It involves securing the applications that run on the cloud servers. This includes securing the code that makes up the application, the data that the application processes, and the users that access the application. Application security is important because it protects the cloud servers from attacks that exploit vulnerabilities in the application code.

So, before approaching the cloud, it is important to understand the security risks and challenges that come with it. The most common security risks and challenges associated with cloud computing are:

- **Data Breaches**: Data breaches are one of the most common security risks associated with cloud computing. A data breach occurs when an unauthorized party gains access to sensitive data stored in the cloud. This can happen through a variety of means, such as hacking, phishing, or social engineering.
- **Data Loss**: Data loss is another common security risk associated with cloud computing. Data loss occurs when data stored in the cloud is accidentally deleted, corrupted, or otherwise lost. This can happen due to a variety of reasons, such as hardware failure, software bugs, or human error.
- **Account Hijacking**: Account hijacking is a security risk that occurs when an unauthorized

party gains access to a user's cloud account. This can happen through a variety of means, such as phishing, social engineering, or weak passwords. Once an attacker gains access to a user's account, they can steal sensitive data, delete files, or perform other malicious activities.

- **Insecure APIs**: Insecure APIs are a security risk that occurs when the APIs used to access cloud services are not properly secured. This can happen due to a variety of reasons, such as weak authentication mechanisms, lack of encryption, or other vulnerabilities in the API implementation. Insecure APIs can be exploited by attackers to gain unauthorized access to sensitive data or perform other malicious activities.
- **Insider Threats**: Insider threats are a security risk that occurs when an authorized user of a cloud service intentionally or unintentionally causes harm to the service. This can happen due to a variety of reasons, such as disgruntled employees, careless users, or users who are tricked by attackers. Insider threats can be difficult to detect and prevent, as the attacker already has legitimate access to the service.

Host Level

- **Hypervisor security**: The hypervisor is a critical component of the cloud infrastructure, as it is responsible for managing the virtual machines that run on the physical servers. Hypervisor security is important because a compromise of the hypervisor can lead to a compromise of all the virtual machines that run on it. Hypervisor security can be improved by using secure hypervisors, keeping the hypervisor up to date, and using strong access controls.
- **Virtual Machines security**: Virtual machines are the building blocks of the cloud infrastructure, as they run the applications that make up the cloud services. Virtual machine security is important because a compromise of a virtual machine can lead to a compromise of the data and applications that run on it. Virtual machine security can be improved by using secure virtual machine images, keeping the virtual machines up to date, and using strong access controls. Also ssh keys can be used to access the virtual machines.

The VM/Containers security has the advantages of a simpler implementation of resource management policies, along with improved intrusion prevention and detection and more efficient software testing.

The OS implement minimal security on: Access control, authentication usage and cryptographic usage policies. Applications with special privilegies that perform security-related functions are called trusted applications. They should only be allowed in the lowest level of privileges required to perform their functions. An OS poorly isolates one application from another, and once an application is compromised, the entire physical platform and all applications running on it can be affected.

For **Data Security**, Identify the security boundary separating the client's and vendor's responsibilities Determine the sensitivity of the data to risk Data should be transferred and stored in an encrypted format. Separate clients from direct access to shared cloud storage.

The following are the mechanism for protecting data:

- **Access control**, which is the process of determining who can access what data and under what conditions. Access control is typically implemented using a combination of authentication, authorization, and auditing mechanisms.
- **Auditing**, which is the process of monitoring and recording access to data. Auditing is typically implemented using a combination of logging, monitoring, and reporting mechanisms.
- **Authentication**, which is the process of verifying the identity of a user. Authentication is typically implemented using a combination of passwords, biometrics, and other authentication mechanisms.

- **Authorization**, which is the process of determining what data a user can access. Authorization is typically implemented using a combination of access control lists, role-based access control, and other authorization mechanisms.

You should isolate data from direct client access, creating a layered access to it. Use data segregation based on tenants.

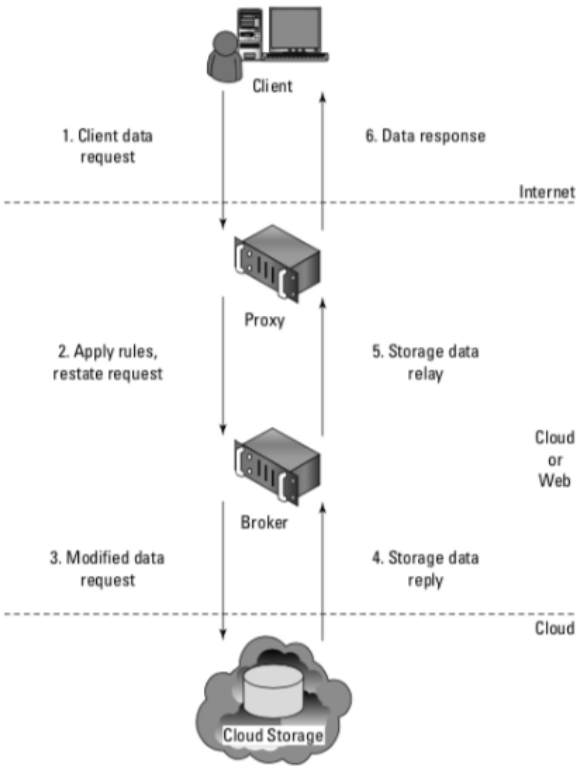


Figure 7.11: Data Security

Most clooud service providers store data in an encrypted form on server side or client side.
Problems:

- a problem with encrypted data may result with data that may not be recoverable.
- it does nothing to prevent data loss: keep your keys!