Cloud Computing

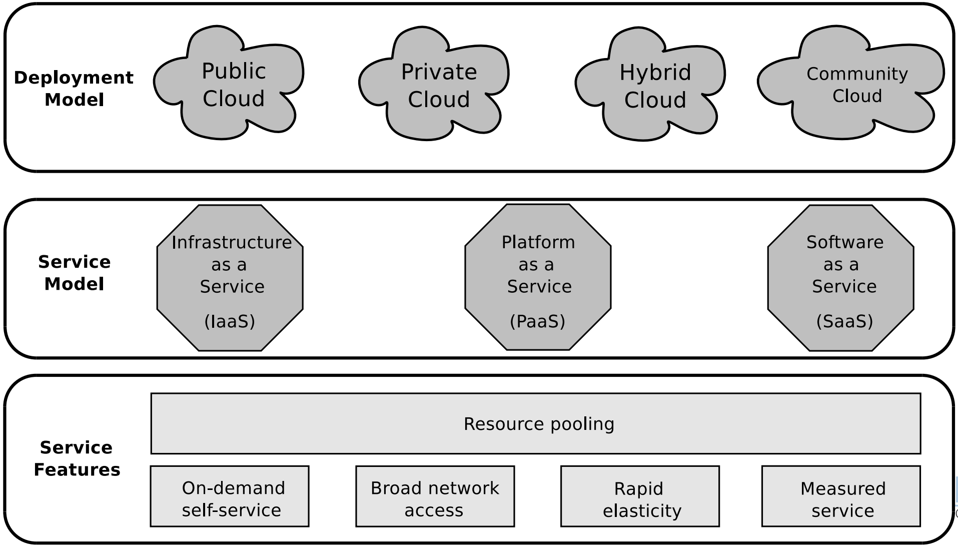
Technological foundations of Cloud Computing
Slide set 2

Henry-Norbert Cocos
cocos@fra-uas.de

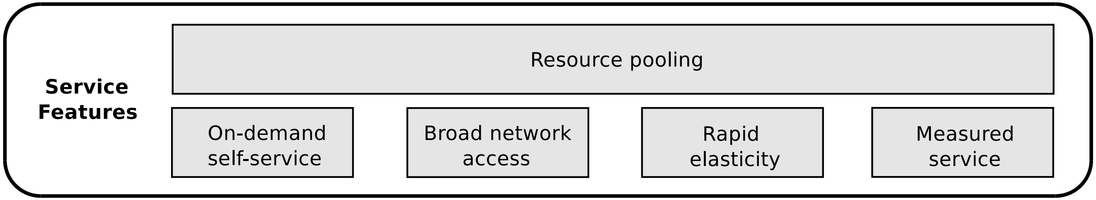
Computer Science
Department of Computer Science and Engineering
Frankfurt University of Applied Sciences

Agenda

NIST definition of Cloud Computing



Service Features



Resource pooling

- Computing resources are pooled to serve multiple consumers.

On-demand self-service

- Consumer can provision computing capabilities automatically.

Broad network access

- Capabilities are available and accessed over the network.

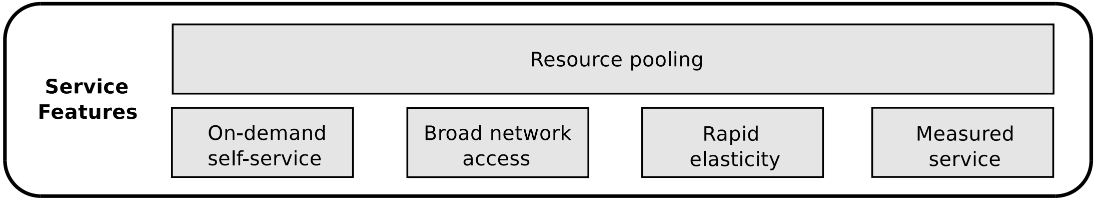
Rapid elasticity

Capabilities can be elastically provisioned to scale with demand.

Measured service

Automatically controlled and optimized resources with metering.

Technological implementation of Service Features



Resource pooling **On-demand self-service** **Broad network access** **Rapid elasticity** **Measured service**

Group Discussion

Do you already know technologies, that offer these services?

Legacy IT

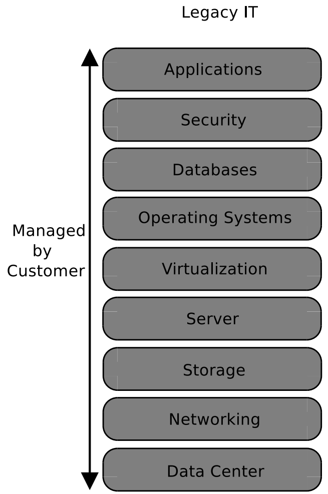


Figure: Layers of operation in legacy IT

Legacy IT

- Data center technology for the operation of applications and storage of data
- Servers for the operation of applications for a business
- Networking technologies for the physical connection of the servers
- Storage capabilities for the storage of large amounts of data

Virtualization

Not part of legacy IT! But helpful for the operation of workloads! (More in slide ?? to ??)

Data center

Source: Adobe Stock

Data center technology

- Enclosure in cabinets (server racks).
- Air conditioning and centralized cooling.
- Electrical power with uninterruptible power supply (UPS).
- Fire protection with sprinklers or gas (Halon gas).
- Security though personal, access control and other features.

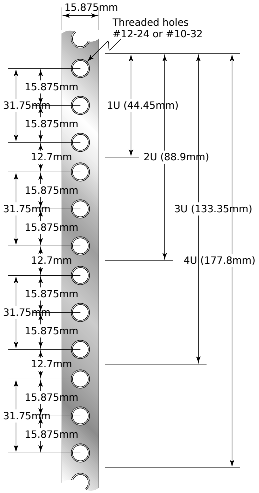
Design of data centers

Data centers are designed to meet high standards on the availability of the services!

FURT
RSITY
OF APPLIED SCIENCES

Server racks

Source: Wikipedia

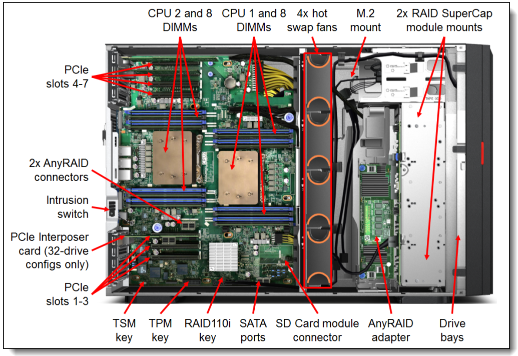


19 inch server rack

- Servers are mounted on rails for easy access.
- A rack consists of networking devices (switches, routers), servers and storage (SAN).
- All components are setup together for simpler operation and maintenance.
- The unit of measurements for components is 1 Unit (44,45mm)
- A standard cabinet has 42 Units for the installation of components

Servers

Source: Lenovo Press



Server grade hardware

- Server grade hardware is designed for 24/7 operation.
- Mainboards ususally contain > 2 CPUs per board and have larger caches.
- RAM sizes are usually very large in size (> 1 TB).
- Network interface cards (NIC) are usually equipped with multiple ports, different port specifications and have high bandwidths (>10 GBit/s!).
- Storage capabilities are usually designed for fault tolerance and performance (e.g. RAID 6).
- Power supplies and generally the hardware designed redundantly (e.g. Battery backed units).

Example Server – ASUS RS700A-E11-RS12U

Source: ASUS

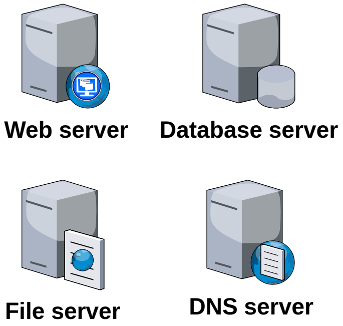
Specifications

- **CPU:** 2x AMD EPYC 7513, 32 Cores, 2.6GHz, 128MB L3 Cache,
- **RAM:** 1024 GB DDR4 3200MHz ECC Reg. (32x 32GB)
- **SSD:** 2x 512GB Kioxia XG6 Series, M.2
- **SSD:** 2x 800GB Micron 7300 MAX, 2.5 inch
- **HDD:** 4x 7,68TB Kioxia CD6-R, 3.5 inch
- **NIC:** 1x Dual Port Intel 10Gbit LAN + 1 x 1Gbit Mgmt LAN
- **NIC:** 1x Dual Port 25GbE Mellanox, SFP28
- **Size:** 1U

Price: approx. 20.000 €

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Server types



Depending on the usage scenario different types specialized for specific fields of application exist!

- **Web server**
  - Designed for the hosting of websites.
  - Supports web standards (HTTP/S) and frameworks (HTML/JavaScript).
- **Database server**
  - Specialized for data storage and retrieval (Relational/NoSQL).
  - Concurrent access of data by users.
- **File server**
  - Serves files or file systems to clients.
  - Supports standard protocols (FTP/S).
- **DNS server**
  - Specialized for the name resolution in computer networks.
  - Functionality by DNS-Software (e.g.`dnsmasq`)

Bare metal servers

Bare metal server

- One standalone physical server
- One dedicated OS installed on the server
- Exclusive access to all hardware resources (CPU, RAM, storage)
- Suitable for high performance, because of direct access to hardware capabilities

Drawbacks

- Management and maintenance is complex
- Management and maintenance is very expensive
- High cost for operation of servers
- Dedicated usage of server leads to many idle times

Solution!

Operation of applications/services in virtual machines (VM)!

Server virtualization and consolidation

- By using an additional abstraction layer (more details in slide **??**) between hardware and software a single physical server allows for the operation of multiple VMs, which in turn allow support for applications that normally require dedicated servers.
- This process is called **server virtualization**, since the offered resources are virtualized!
- This way resources from a single server can be shared among multiple virtual machines, reducing the number of necessary server hardware!
- The reduction of servers by using virtual machines for applications is called **server consolidation**!

Benefits

- **Increased Efficiency:** Consolidating multiple physical servers onto fewer hardware resources improves overall resource utilization.
- **Improved Resource Utilization:** Server consolidation allows for better utilization of existing resources, as virtual machines can dynamically receive resources based on their current demand.
- **Cost Savings:** Fewer physical servers mean reduced hardware costs, lower energy consumption, less data center space required, and lower overall operational expenses.
- **Simplified Management and Scalability:** Virtual machines can be deployed, moved, and scaled quickly as needed, increasing flexibility and responsiveness to changing demands.
- **Enhanced Fault Tolerance and Disaster Recovery:** Consolidating onto fewer hardware nodes makes it easier to implement redundancies and improve fault tolerance.

Overall, server consolidation can improve the efficiency, flexibility, and cost-effectiveness of an IT infrastructure while simplifying management and optimizing performance.

Virtual servers dimensioning – rough estimates

For the estimation of dimensioning virtual servers for the operation on physical hardware servers the following rules of thumb can be used.

1. **CPU Core Ratio:**

$$\text{Maximum number of VMs} = \frac{\text{Number of CPU cores on the server}}{\text{Number of CPU cores per VM}}$$

2. **RAM Ratio:**

$$\text{Maximum number of VMs} = \frac{\text{Available RAM on the server}}{\text{RAM per VM}}$$

3. **Storage Space Ratio:**

$$\text{Maximum number of VMs} = \frac{\text{Available storage space on the server}}{\text{Storage space per VM}}$$

Overprovisioning of resources

Usually a VM is not allocating the complete amount of physical resources assigned. Therefore servers are usually overprovisioned and the ratio of vCPU to (physical) CPU cores is in the range of 2:1 up to 5:1 depending on the use case.

Virtual servers dimensioning – example

For the ASUS RS700A server from slide **??** we get the following estimates:

No Overprovision **1:1**

[a] **CPU Core Ratio:**

$$\textbf{Max 32 VMs} = \frac{\text{64 pCPU cores}}{\text{2 vCPU cores per VM}}$$

RAM Ratio:

$$\textbf{Max 128 VMs} = \frac{\text{1024 GB physical RAM}}{\text{8 GB per VM}}$$

Storage Space Ratio SSD:

$$\textbf{Max 16 VMs} = \frac{\text{1600 GB}}{\text{100 GB per VM}}$$

Storage Space Ratio HDD:

$$\textbf{Max 314 VMs} = \frac{\text{31.457 GB}}{\text{100 GB per VM}}$$

[a] As a reference for calculation the specs of the VMs are all equal!

NKFURT
VERSITY
SCIENCES

Virtual servers dimensioning – example with overprovisioning

For the ASUS RS700A server from slide **??** we get the following estimates:

Overprovisioning	**CPU Core Ratio:**	**RAM Ratio:**	**Storage Space Ratio SSD:**	**Storage Space Ratio HDD**[1]**:**
1:1	Max. 32 VMs	Max. 128 VMs	Max. 16 VMs	Max. 314 VMs
2:1	Max. 64 VMs	Max. 256 VMs	Max. 32 VMs	Max. 628 VMs
3:1	Max. 96 VMs	Max. 384 VMs	Max. 48 VMs	Max. 942 VMs
4:1	Max. 128 VMs	Max. 512 VMs	Max. 64 VMs	Max. 1256 VMs
5:1	Max. 160 VMs	Max. 640 VMs	Max. 80 VMs	Max. 1570 VMs

Overprovisioning for use cases

For CPU intensive tasks (e.g. databases) the amount of overprovisioned resources should not be to high! A ratio of **2:1** should therefore not be exceeded! For simpler workloads with more idle process usage a ratio of **3:1** is possible. Ratios above **4:1** could lead to issues and performance degradation!

[1]This value is misleading! But the purpose of backups and snapshots is reasonable.

From traditional IT to the cloud...

We have investigated data centers and servers for the operation in traditional IT. This sets the foundation for the installation and operation of cloud services.

In the next section we will investigate the technologies needed for the operation of cloud services being:

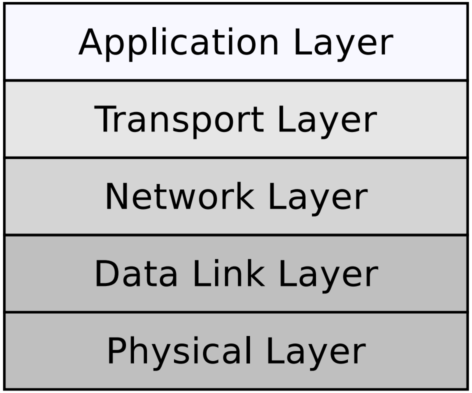
- Networking in data centers
- Storage in data centers
- Virtualization technologies
- How to measure service offerings from provider site

Only a brief overview ;-)

We do not have the time to discuss the mentioned technologies in detail. Yet it is very important to get an overview of the used technologies that enable Cloud Computing offers. Therefore we will take a very brief look at these technologies!

TCP/IP networks

Hybrid Reference Model



Layer 5 **Application Layer**
- **Purpose:** Definition of standards for applications and interfacing
- **Protocols:** HTTP/S, FTP/S, SSH

Layer 4 **Transport Layer**
- **Purpose:** Flow control and segmentation of data
- **Protocols:** TCP, UDP QUIC

Layer 3 **Network Layer**
- **Purpose:** Addressing, routing and packing (logical) IP-packets
- **Protocols:** IPv4, IPv6

Layer 2 **Data Link Layer**
- **Purpose:** Addressing and packing (physical) Ethernet frames and media control
- **Protocols:** Ethernet

Layer 1 **Physical Layer**
- **Purpose:** Encoding data on physical media
- **Protocols:** Signals

Networking in data centers – very brief overview ;-)

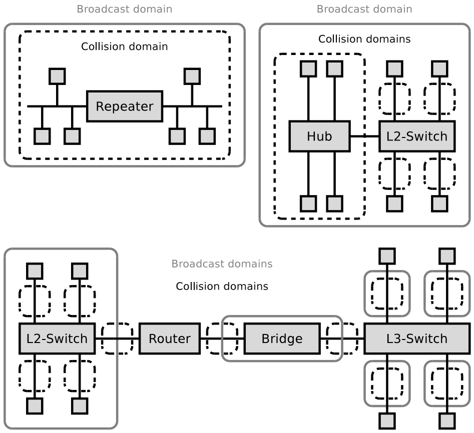
- **Addressing**
  - MAC-addresses for physical addressing of devices and appliances
  - 48 bit for the addressing of devices (e.g. `38:f3:ab:d2:34:67`)
  - IP-addresses for logical addressing servers and appliances
  - With IPv4 a maximum of 4.294.967.295 devices can be addressed
  - With IPv6 a maximum of 3.4×10^{38} devices can be addressed
- **Private LAN**
  - Private network area for local machines (e.g. data center)
  - Switched networks seperate collision domains
  - Messages internally are not routed (Broadcast Domain)

- **IP Networks and their limitations**
  - Physical networks cannot be subdivided properly and therefore extend the broadcast domain!
  - IP-addresses are limited and routing is not necessary in LAN!
  - Administration of physical and logical networks is very complex and rigid!
  - Shared transmission medium for different traffic is not ideal!

Collison and Broadcast domain

Collision domain:
A collision domain is an area of a network in which collisions between data packets can occur. In Ethernet networks based on CSMA/CD (Carrier Sense Multiple Access with Collision Detection), collisions can occur when two devices attempt to send data via the same transmission medium at the same time.
Broadcast domain:
A broadcast domain is an area of a network in which a broadcast packet is sent and received by all devices in that area. Broadcast packets are messages that are sent to all devices in the network without having a specific destination address.

Networking in data centers – devices



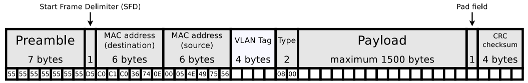
Collision and Broadcast Domains

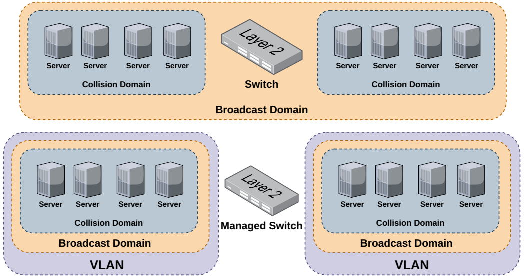
- **Layer-2-Switches**
  - State of the art in LAN
  - Separate Collision Domains of Ethernet networks
  - Act on Data-Link-Layer without addresses
- **Layer-3-Switches and Routers**
  - Manage logical networks and use IP-addresses
  - Separate Broadcast Domains of logical IP-based networks
  - Function as gateways between different logical (IP) networks

Problem

The subdivision in this form is very inflexible! Since servers inside a data center are inside of (very large) LANs we need a method to subdivide the LAN (and broadcast domain) on the physical Data-Link-Layer without the use of IP-addresses!

Virtual LAN





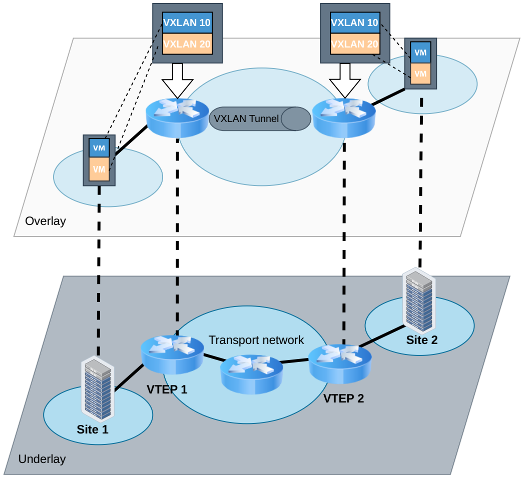
VLAN Benefits

- **Segmentation:** VLANs allow a physical network to be divided into multiple logical networks.
- **Security:** By splitting traffic across different VLANs, administrators can increase security by restricting access to sensitive data or resources.
- **Security:** By splitting traffic across different VLANs, administrators can increase security by restricting access to sensitive data or resources.
- **Reduce broadcast domains:** In a flat network, broadcast messages sent from one device can reach all devices on the network and put a strain on network resources. By segmenting the network into VLANs, broadcast domains can be reduced, resulting in more efficient use of network bandwidth.
- **Flexibility:** VLANs allow administrators to create logical groups of devices that can communicate with each other regardless of their physical location on the network.
- **Performance optimization:** By segmenting the network into VLANs, administrators can optimize network performance by better controlling and prioritizing traffic within the network.

Limitations

VLAN can only address 4094 distinct physical networks. This may be enough for hardware servers but not enough for VMs!

Overlay networks



Overlay networks

- Separation of physical network (so called underlay network) from logical networks (so called overlay networks).
- Network traffic can be transmitted via a physical transport network and routed through different intermediate stations.
- The logical traffic (e.g. from VMs) is tunneled and therefore the transport is transparent to the endpoints
- Geographically distributed sites can be connected via a tunnel and form large logical LAN

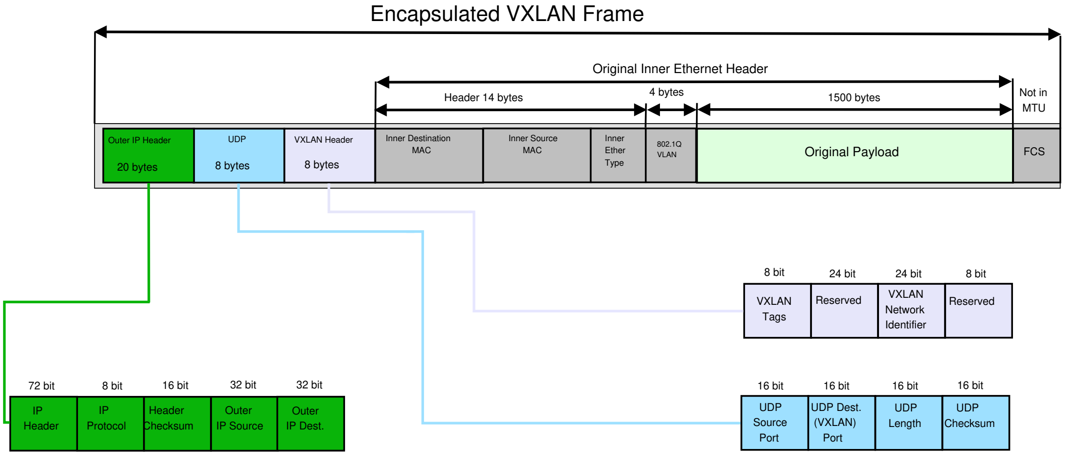
VXLAN

- Protocol for the overlay connection of physical LAN over UDP.
- VXLANs can be addressed using 24 bit VXLAN Network Identifier (VNI).
- The VNI can be used to establish a VXLAN Tunnel Endpoint (VTEP) for the connection of different physical sites.
- VXLAN uses UDP as a transport protocol and tunnels ethernet frames in UDP segments (see slide ??).

Number of VXLAN addresses

VXLAN is an extension of VLAN. In contrast to VLAN VXLAN can address up to 16 million distinct isolated VXLAN. This is enough for VMs!

VXLAN Header



Benefits of VXLAN compared to VLAN

- **Scalability:** VLANs have a limited number of IDs (4096), which can quickly be exhausted in large data centers with many virtual machines (VMs) or containers. VXLAN uses a 24-bit VXLAN segment ID, theoretically allowing for over 16 million isolated networks. This makes VXLAN better suited to meet the demands of large and constantly changing data center environments.
- **Multi-Tenancy Support:** VXLAN provides better support for multi-tenancy scenarios where different customers or applications require isolated networks. With VXLAN, multiple virtual networks can run over the same physical network segment, improving resource utilization and enhancing security and isolation.
- **Flexibility and Mobility:** VXLAN enables the virtualization of Layer-2 networks across Layer-3 networks. This allows VMs or containers to be moved between different physical locations without needing to change their IP addresses. This enables applications and workloads to scale or move flexibly and without interruption.
- **Integration with SDN and Cloud Technologies:** VXLAN is often tightly integrated with Software-Defined Networking (SDN) and cloud technologies, facilitating the automation, orchestration, and management of networks in dynamic and virtualized environments.

VXLAN in data centers

VXLAN (Virtual Extensible LAN) offers several advantages over traditional VLANs (Virtual Local Area Networks), especially in data center environments. Overall, VXLAN provides a powerful solution for scaling, isolating, and providing flexibility to networks in modern data center environments compared to traditional VLANs.

Storage in data centers – very brief overview ;-)

Servers operated in data centers also have a huge demand for storage! We can distinguish storage devices by their access level to data:

- **File-level access** – Access on storage medium on file system level on a directory and file level.
- **Block-level access** – Access on storage medium on block level as smallest addressable unit

Block-level storage or block storages

We focus on Block-level storage and is also shortened block storage, since this type of storage is needed to define file systems. For the operation of virtual machines block-level storage is needed to operate the virtual machines' hard drive!

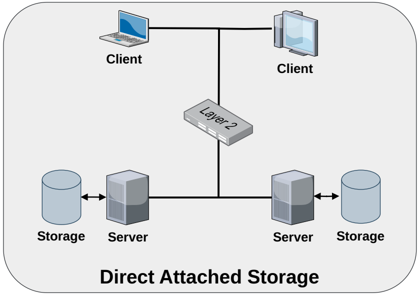
Offering block storage in data centers can be done with different technologies:

- **Direct attached storage (DAS)**
- **Network attached storage (NAS)**
- **Storage Area Networks (SAN)**
- **Software-defined Storage (SDS)**

Only a brief overview ;-)

We will not dive into these technologies but we will take a brief look at these technologies and their characteristics.

Direct Attached Storage



Direct Attached Storage

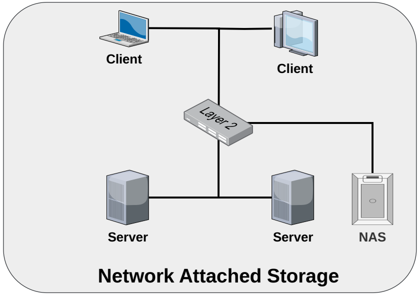
- Direct connection to servers through Host Bus Adapter (HBA)
- Usually setup as JBOD (Just a Bunch of Disks)
- RAID setups are also possible
- Highly scalable with up to 100 drives
- Protocols: NFS, CIFS, iSCSI

Strengths and weaknesses

+ Low hardware costs
+ Low costs
+ No additional protocol stack
+ Conceptually very performant

- Exclusive to one host
- Limited scalability in capacity
- Limited distance from host to storage

Network Attached Storage



Network Attached Storage

- Connection of NAS via standard ethernet switches
- Simple administration and easy integration into network infrastructure
- High bandwidths are also possible ($\geq$ 10Gbit/s)
- Networking-Filesystem (SMB/CIFS, NFS)

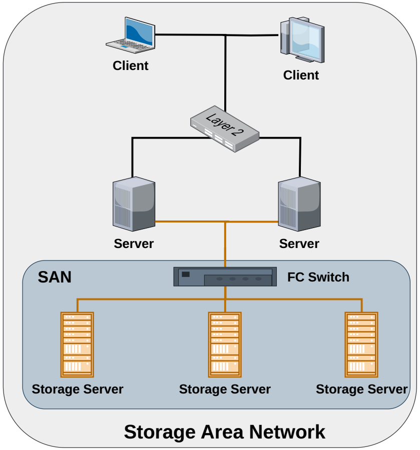
Strengths and weaknesses

+ Simple connection to existing network
+ With dedicated 10 Gbit/s LAN in real operation true alternative to FC-SAN
+ Easy concurrent access for multiple clients thanks to networking file system

- Underlying TCP/IP protocol not optimized for storage traffic
- Additional high load on the existing LAN

FURT
RSITY
NCES

Storage Area Networks



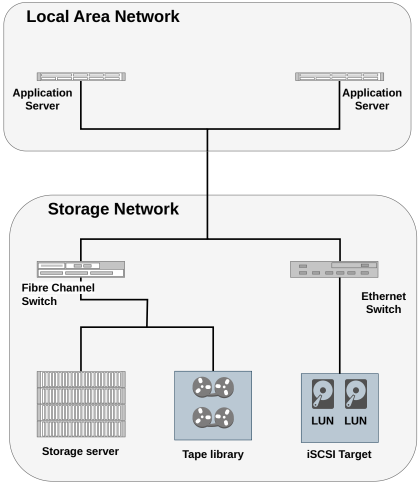
Storage Area Networks

- Attached via Fibre-Channel-HBA and FC-Switch
- Setup of storage in own network infrastructure
- Very high bandwidths over Fibre-Channel
- Centralized administration for storage network
- Highly scalable with up to 100 drives

Strengths and weaknesses

+ Direct accessibility of all components to each other	- Additional dedicated infrastructure level for storage
+ High transfer rates	- Problems with interoperability
+ Cost savings through storage consolidation	- Expensive hardware components
+ Easier administration through centralization	- Complicated configuration

Storage Area Networks in data centers



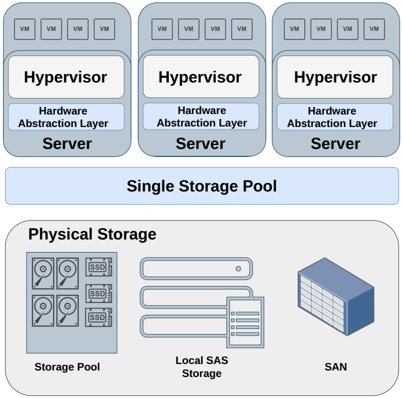
Storage Area Networks

- **SAN over Fibre Channel**
  - Block-level access
- **Tape libraries**
  - Backup of data on analogue tape
- **iSCSI Target Server**
  - iSCSI block-level access over TCP
  - Logical unit numer (LUN)

Advantage

The most significant advantage of a storage area network is the direct accessibility of all components integrated into the SAN to each other Cluster applications are easy to implement in the storage area network; every server can access every block device (LUN), and concurrent write access is even possible with the right software

Software Defined Storage



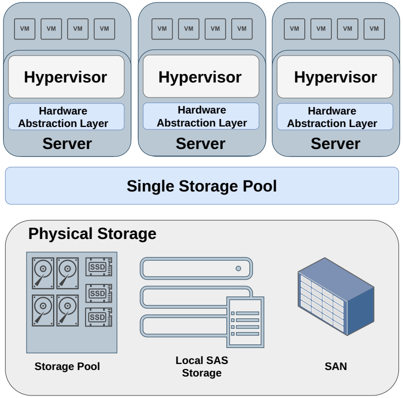
Software defined Storage

- **VM → Hypervisor**
  - VMs run on Hypervisor and access storage through virtual storage definition
- **Single Storage Pool**
  - Storage is defined logically by software and served as one unified storage pool
- **Physical storage**
  - Physical devices for reading and storing data
  - Different device types can be mapped by software to abstract vitual storage devices

Advantage

The most significant advantage of a software defined storage is the abstraction of physical storage by the abstraction layer. This improves the management and flexible attachment of physical storage to virtual machines!

Software Defined Storage



Benefits

- **Flexibility and Scalability:** SDS abstracts storage resources from the underlying hardware, allowing for greater flexibility and scalability. Administrators can dynamically allocate and reallocate storage capacity as needed, without being tied to specific hardware configurations.
- **Simplified Management:** SDS platforms typically provide centralized management interfaces that enable administrators to easily provision, monitor, and manage storage resources from a single dashboard.
- **Automation and Orchestration:** SDS platforms often include features for automation and orchestration, allowing organizations to streamline storage provisioning, data migration, and other storage-related workflows.
- **Scalability Across Environments:** SDS solutions are designed to scale across diverse environments, including on-premises data centers, hybrid cloud deployments, and multi-cloud environments.
- **Improved Performance and Availability:** Many SDS platforms include features for data replication, snapshotting, and other data protection mechanisms that improve data availability and resilience. Additionally, SDS allows organizations to leverage advanced storage technologies, such as flash storage and NVMe, to achieve higher levels of performance and responsiveness for critical workloads.

Virtualization – Fundamentals and general idea

- By using virtualization, the resources of a computer system can be split and used by multiple independent operating system instances
- Several fundamentally different approaches and technologies exist to implement virtualization
- Each virtual machine (VM). . .
  - behaves like any other computer, with own components
  - runs inside an isolated environment on a physical machine
- Inside a VM, an operating system with applications can be installed, exactly like on a physical computer
- The applications do not notice that they are located inside a VM
- Requests from the operating system instances are transparently intercepted by the virtualization software and converted for the existing physical or emulated hardware
- The VM itself does not become aware of the virtualization layer between itself and the physical hardware

Virtualization concepts

History of Virtualization

- 1970/71: IBM introduced the Virtual Machine Facility/370 (VM/370)
- On this platform, multi-user operation is implemented by using multiple single-user mode instances, which are executed in virtual machines
- Each VM is a complete duplicate of the underlying physical hardware

Creasy RJ. **The origin of the VM/370 time-sharing system.** IBM Journal of Research and Development (1981), No. 5, P.483–490,

Different virtualization concepts exist:

- Partitioning
- Hardware emulation
- Application virtualization
- **Full virtualization (Virtual Machine Monitor = Type-2 Hypervisor)**
- **Paravirtualization (Type-1 Hypervisor)**
- Hardware virtualization
- **Operating system-level virtualization / Container / Jails**
- Storage virtualization (SAN/SDS)
- Network virtualization (VLAN/VXLAN)

Hardware Virtualization

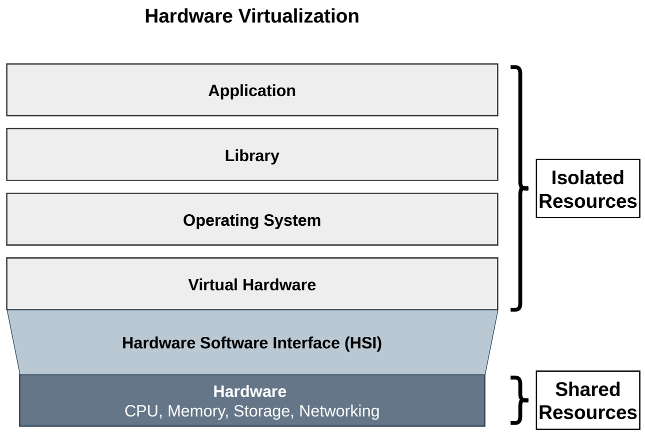


Figure: Hardware Virtualization

Hardware Virtualization

- Hardware resources (CPU, RAM, Networking, etc.) are shared between different OS instances
- The Virtual Machine Monitor translates the requests between the Hardware and the OS
- The OS instances are isolated from each other
- The OS is running on virtualized hardware resources which are mapped to physical resources
- Different types of virtiualization exist depending on the level of abstraction

Virtualization Basics of the x86 Architecture

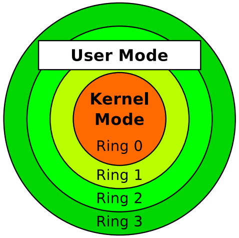
- x86-compatible CPUs implement 4 privilege levels
  - Objective: Improve stability and security
  - Each process is assigned to a ring permanently

- Implementation: The register CPL (Current Privilege Level) stores the current privilege level
- Source: `http://css.csail.mit.edu/6.858/2012/readings/i386.pdf`

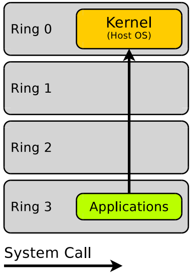
- In ring 0 (= **kernel mode**) runs the kernel
  - Processes here have full hardware access
- In ring 3 (= **user mode**) run the applications
  - Processes are in Protected Mode

Unmodified modern operating systems only use 2 privilege levels (rings) $\Longrightarrow$ Rings 0 and 3

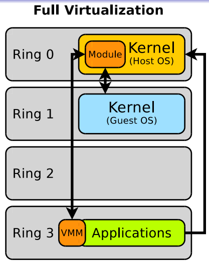
- If a user-mode process must carry out a higher privileged task (e.g. access hardware), it can tell this the kernel via a **system call**
  - The user-mode process generates an exception, which is intercepted in ring 0 and handled there



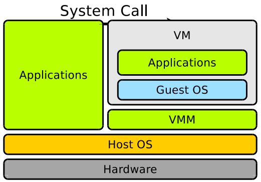
Without Virtualization



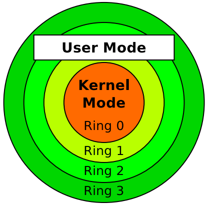
Full Virtualization (2/3)



- Full virtualization makes use of the fact, that x86 systems typically use only 2 privilege levels
  - The VMM runs together with the applications in ring 3
  - VMs are in the less privileged ring 1



- The VMM contains for every exception a treatment, which catches, interprets and executes privileged operations of guest operating systems



- VMs can only access the hardware via the VMM
  - This ensures controlled access to shared system resources

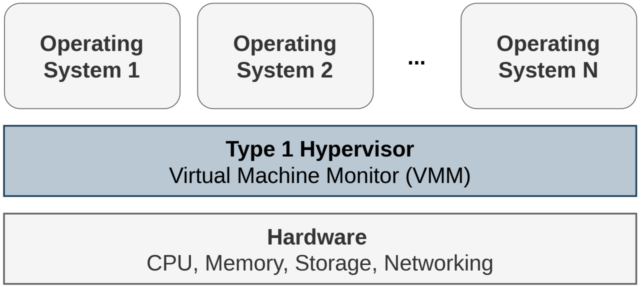
Full Virtualization (3/3)

- Advantages:
  - Few modifications in the host and guest operating systems are required
  - Access to the main resources is just forwarded (passed through)
    $\Longrightarrow$ guest operating systems run almost with native performance
  - Each guest operating system has its own kernel
    $\Longrightarrow$ high degree of flexibility
- Drawbacks:
  - Switching from one ring to another one requires a process/context switch
    $\Longrightarrow$ each process/context switch consumes CPU time
  - If an application in the guest operating system requests the execution of a privileged instruction, the VMM provides a replacement function, which commands the execution via the kernel API of the host operating system
    $\Longrightarrow$ speed losses

Full Virtualization Examples: Some virtualization solutions, which implement the VMM concept

- VirtualBox
- VMware Server, VMware Workstation and VMware Fusion
- Parallels Desktop and Parallels Workstation

Type-1 Virtualization – Hypervisor



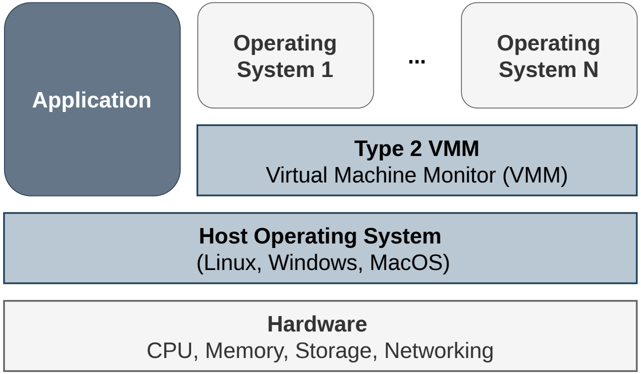
Type 1 Virtualization

- The Hypervisor runs between the hardware and the operating system(s)
- The Hypervisor acts as a (specialized form of) operating system
- The VMs run as processes on the Hypervisor
- The VMs are unaware of this fact
- The VM need drivers for the interaction with the Hypervisor
- Also called bare-metal Hypervsior

Examples

VMWare vSphere, Microsoft Hyper-V, Kernelbased Virtual Machine (KVM)

Type-2 Virtualization – Virtual Machine Monitor



Type 2 Virtualization

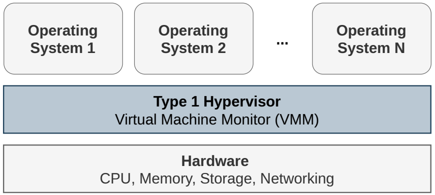
- The VMM runs on an operating system
- The VMM is another process inside of the operating system and runs among other applications
- The VMM has lower privilege than the host operating system

Examples

Oracle Virtualbox, VMWare WorkStation,MacOS Parallels

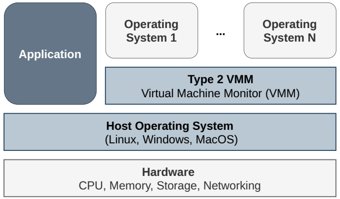
Comparison of Virtualization types

Type-1 Virtualization



- Approx. 5% performance overhead compared to physical (bare metal) operation
- Resources are managed directly by Hypervisor
- More complex administration skills necessary
- More isolation of virtual environments

Type-2 Virtualization



- Approx. 10% performance overhead compared to physical (bare metal) operation
- Runs inside of host OS and therefore just indirect management of resources
- Easier to administer inside of host OS
- Less isolation than virtual environments

Cloud providers

Because of the performance benefits CSP use Type-1 Hypervisors!

Operating System Virtualization

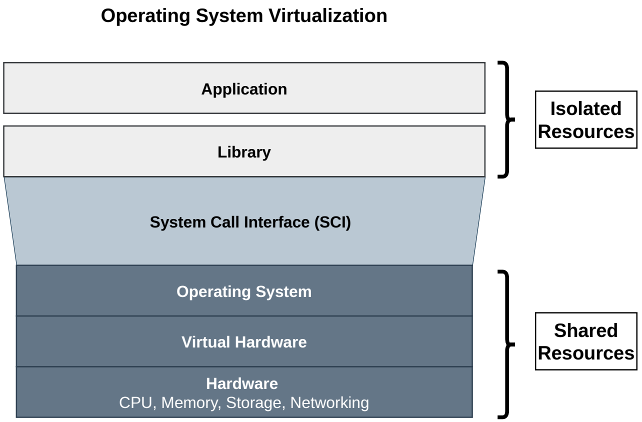
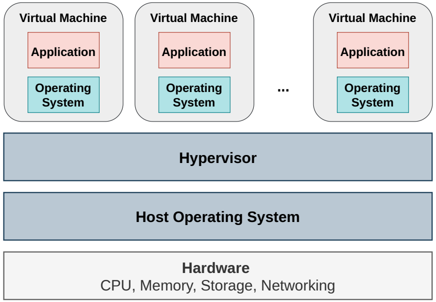


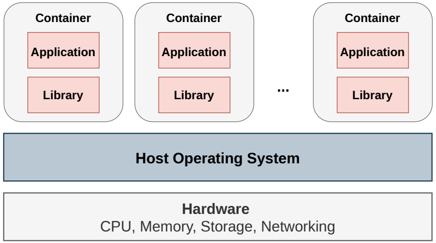
Figure: OS Virtualization

Operating System Virtualization

- Operating System virtualization does not use a Hypervisor between the hardware and the virtual environment
- The virtual environments run on a host OS and share the OS resources and underlying hardware resources
- The System Call Interface is managing the requests of the virtual environments and handles the hardware resources
- The virtual environments are called containers

Hardware vs Operating System Virtualization





HW virtualization

- Operation of full OS inside of VM → **heavyweight**
- More isolation of applications through Hypervisor → **more secure**
- Longer statup times of VMs → **slower**

OS virtualization

- Container only packages libraries and applications → **lightweight**
- Less isolation of applications because of missing Hypervisor → **less secure**
- Lower statup times of VMs → **faster**

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

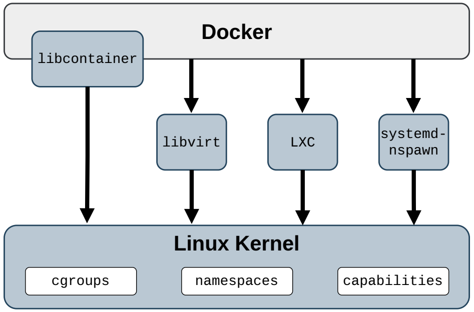
Docker

Docker is a popular platform used for containerization, allowing developers to package applications and their dependencies into isolated containers.

Characteristics

- **Containerization:** Docker enables the creation and management of containers, which are lightweight, portable, and self-sufficient environments that encapsulate an application and its dependencies.
- **Portability:** Docker containers can run on any machine that supports Docker, regardless of the underlying operating system or infrastructure. This makes it easier to develop, deploy, and scale applications consistently across different environments.
- **Isolation:** Containers provide process and resource isolation, ensuring that applications running within them do not interfere with each other.
- **Versioning and image-based deployment:** Docker uses a layered filesystem and Docker images to capture the application's environment and dependencies.

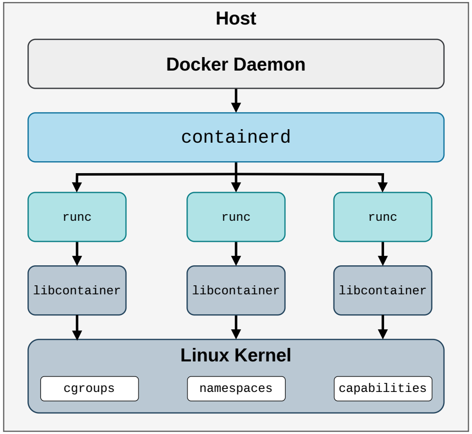
Docker Architecture till version 0.9



Docker Architecture `v.0.9`

- `libcontainer` as a native Go abstraction layer implementation for creating containers
- `libvirt` as an API for the management of containers (also used by other Hypervisors e.g. KVM)
- `LXC` (Linux Containers) as a native Container execution platform
- `systemd-nspawn` to run a command in a light-weight namespace container with virtual file system and IPC
- Linux Kernel functionalities:
  - `cgroups`
  - `namespaces`
  - `capabilities`
  - other libraries for networking, etc.

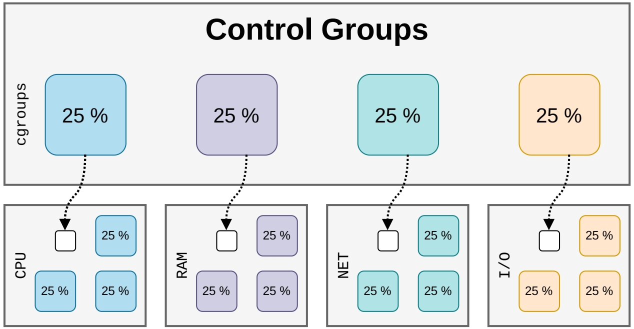
Docker Architecture



Docker Architecture now

- Replaced `LXC` with a new high-level container interface called `containerd`
- `containerd` pulls images, manages networks and storage and uses `runc` to execute containers
- Replaced `libvirt` with a new unified low-level runtime called `runC`
- `runC` uses `libcontainer` for the interaction with Linux Kernel components `runC` offers. . .
    - native support for Linux Kernel security features
    - full support for the complete Linux Kernel namespace

Control Groups



Limits and isolates the resource usage (CPU, memory, disk I/O, etc.) of processes.

Control Groups

- **Resource Limiting:** Sets quota limits for resources like CPU, RAM size, I/O bandwidth limits.
- **Prioritization:** Management of allocation and assignment of resources to processes.
- **Accounting:** Measurement of a group's resource usage and logging of metrics.
- **Control:** Controlling groups and checkpointing and restarting the processes.
- Kernel provides access to multiple controllers for accessing the control groups

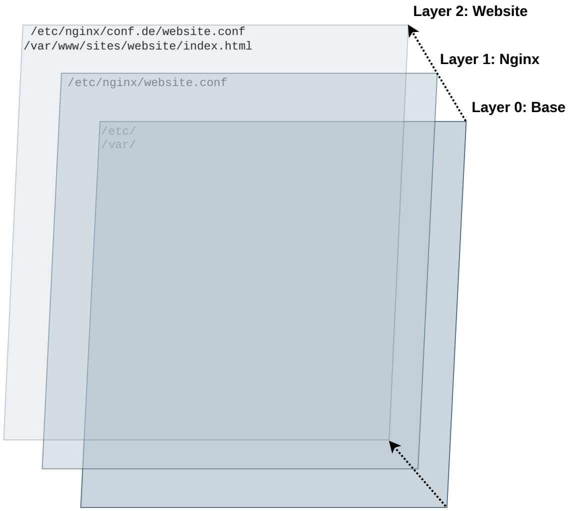
Namespaces

- Linux Namespaces partition the Kernel resources for processes and limits the visibility to processes.
- Namespaces refer to distinct resources and processes can be assigned to one or multiple namespaces
- Examples for namespaces are:
  - **Process ID** (`pid`) – provides processes with an independent set of `pids`
    - `pids` are nested inside namespaces for processes
    - Terminating the first process in a tree will terminate all subsequent processes
  - **Network** (`net`) – virtualizes the network stack and upon creation only the `loopback` interface is present
    - Each namespace has a private set of IP addresses, own routing table, socket listing and other network-related resources.
  - **Interprocess communication** (`ipc`) – isolates the processes' ipc namespaces
    - Produces distinct regions for ipc functions and two processes can use the same identifier across different namespaces

Capabilities

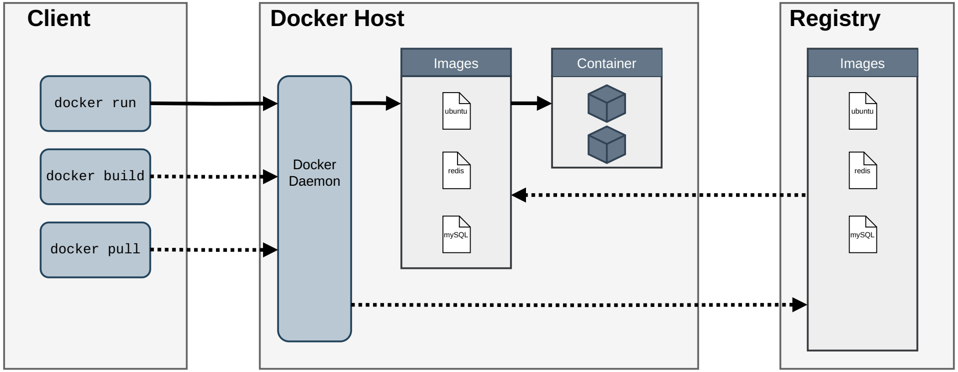
- Linux has two basic controls for privileges:
  - **Superuser** (`root`) – User ID (`uid`) of 0
  - **Normal-user** – non 0 `uid` signals non-root user
- Linux capabilities are used to assign privileges to processes and threads
- Capabilities make it possible to restrict privileges without having full access to the system
- Containers start with a set of capabilities and the Linux Kernel can restrict the access

Union Filesystem



- UnionFS is a layered file system and allows for „*branching*“
- Branches can be transparently overlaid on top of read-only branches
- This way a coherent file system can be presented and used
- Branches are prioritized on mounting and name conflicts can be resolved
- This mechanism is also used in Live CDs

Docker Image Registry

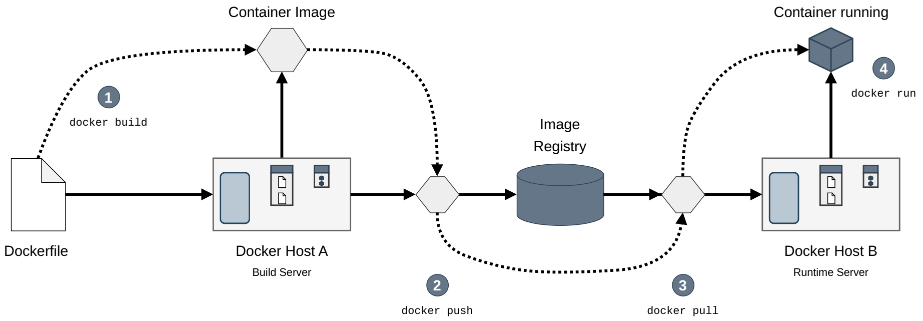


- Open source implementation of a repository for storing and distributing container images and other content
- Two types of registries exist:
  - **Private:** Managed by an institution
  - **Public:** Accessible by public for up- and download
- Natively integrated into the Docker platform

Docker commands

- `docker run` – run containers
- `docker build` – build image and make runnable container
- `docker pull` – pull an image form the registry

Docker Image Workflow



1. Building Docker image from Dockerfile on Host A (Build Server)
2. Pushing image to registry after successful build
3. Docker Host B (Runtime Server) pulls image form registry
4. Docker Host B runs container

This demonstrates the workflow of a container platform. This workflow automates the build process of container images and sets the basis for DevOps Software engineering processes.

NKFURT
VERSITY
OF APPLIED SCIENCES

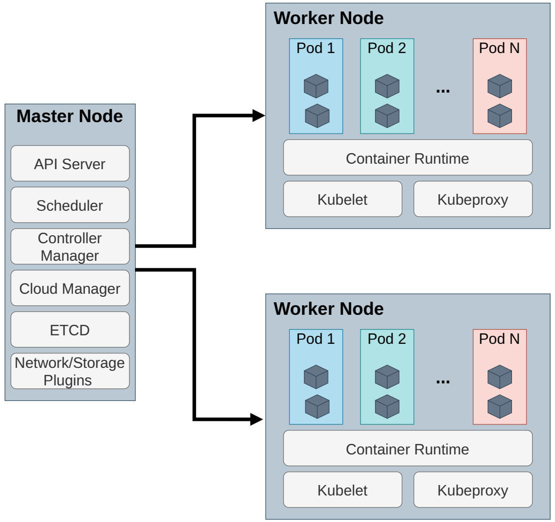
Kubernetes

Kubernetes (also called K8S) is an open-source container orchestration platform designed to automate the deployment, scaling, and management of containerized applications. It provides a highly flexible and resilient infrastructure for deploying and managing distributed systems across clusters of machines.

Characteristics

- **Container Orchestration:** automates the deployment, scaling, and management of containerized applications, handling tasks like scheduling, load balancing, and resource allocation across clusters of machines.
- **Scalability and Flexibility:** allows for horizontal scaling of applications, dynamically allocating resources based on demand, and supporting both stateless and stateful workloads.
- **Self-Healing:** continuously monitors the health of applications and automatically restarts or reschedules containers that fail, ensuring high availability and reliability.
- **Service Discovery and Load Balancing:** It provides built-in mechanisms for service discovery and load balancing, allowing containers to communicate with each other and distributing traffic across multiple instances of an application.
- **Declarative Configuration:** Kubernetes uses a declarative approach to configuration, allowing users to specify the desired state of their applications and infrastructure using YAML or JSON files, which Kubernetes then reconciles with the actual state.
- **Portability and Vendor-agnosticism:** is platform-agnostic and can run on any infrastructure, including on-premises data centers, public clouds, and hybrid environments.

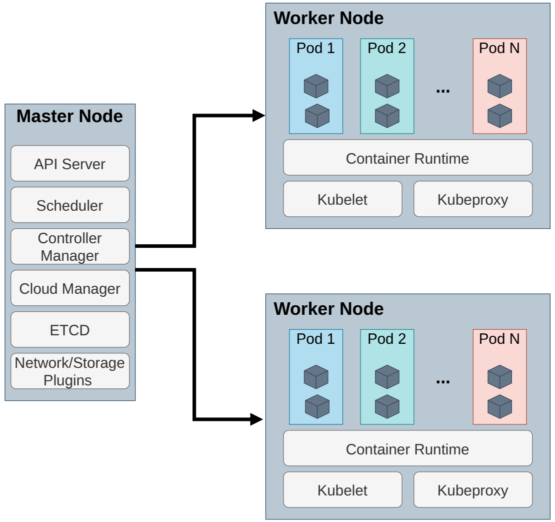
Kubernetes Architecture (1/3)



Master Node

- **API Server:** The Kubernetes API server is the central management component that exposes the Kubernetes API, which clients use to interact with the cluster. It serves as the front-end for the Kubernetes control plane.
- **Scheduler:** The Kubernetes scheduler is responsible for placing newly created pods (groups of containers) onto nodes in the cluster.
- **(Kube) Controller Manager:** The controller manager is a collection of controller processes responsible for managing various aspects of the cluster's state. These controllers include the node controller, replication controller, endpoint controller, and service account controller, among others.

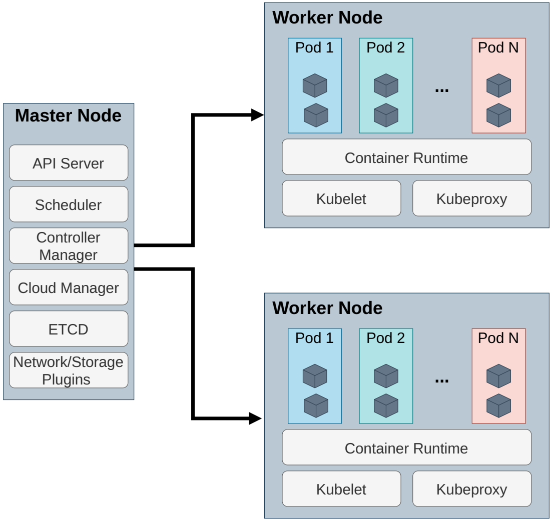
Kubernetes Architecture (2/3)



Master Node

- ***Cloud Manager (optional):*** This component interacts with the underlying cloud infrastructure provider to manage resources such as virtual machines, load balancers, and storage volumes.
- **ETCD:** etcd is a distributed key-value store used by Kubernetes to store cluster configuration and state information. It acts as the cluster's database, storing information such as configuration settings, metadata, and the current state of the cluster.
- **Network/Storage Plugins:** Plugins for the use of network and storage functionalities inside of pods.

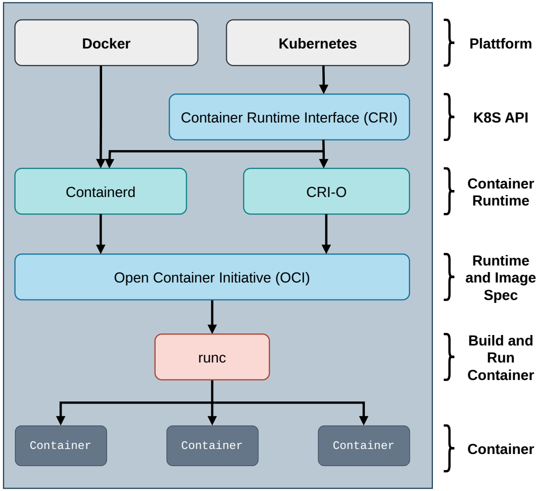
Kubernetes Architecture (3/3)



Worker Node

- **Kubelet:** Primary Node agent for the registration with the API Server.
- **Kubeproxy:** proxy is responsible for network communication within the cluster. It maintains network rules and routes traffic to the appropriate containers or services based on IP address and port number.
- **Pod:** Collection of one or more Containers.

Container Runtime Interface and Open Conainer Initiative



CRI – High-Level Interface

API between Kubernetes and the container runtime

- **Containerd**
  - container runtime
  - `containerd` has a CRI plugin by default (from version 1.1). It allows to run Kubernetes with `containerd` as container runtime
- **CRI-O**
  - open source alternative to `containerd`

OCI – Low-Level Interface

Defines a standard for images and container

- **runC**
  - Low-level container runtime

Measuring service offerings

For the optimal operation of servers and applications measuring these instances is important! Monitoring tools are used to observe, track, and manage the performance, health, and availability of IT infrastructure components, applications, and services.

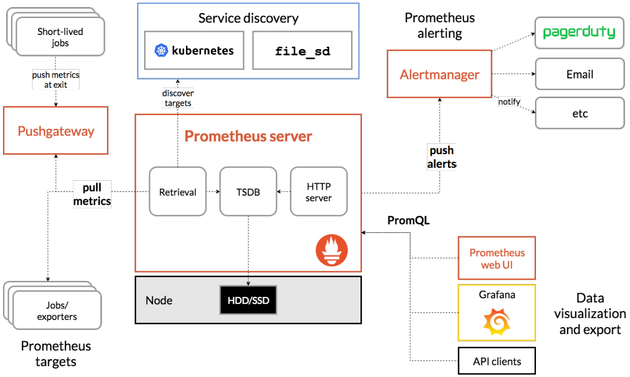
Characteristics

- **Performance Optimization:** provide insights into the performance of IT infrastructure components, applications, and services.
- **Capacity Planning:** help organizations forecast future resource requirements and plan for capacity expansion.
- **Fault Detection and Troubleshooting:** enable organizations to quickly detect and diagnose issues within their IT infrastructure, applications, and services.
- **Compliance and Reporting:** help organizations ensure compliance with regulatory requirements and industry standards by monitoring and reporting on key performance metrics.
- **Service Level Agreement (SLA) Management:** allow organizations to monitor and enforce service level agreements (SLAs) by tracking key performance indicators, uptime guarantees, and response times.

We will look at three monitoring solution that are widely used in the industry.

Prometheus

Source: https://prometheus.io/docs/

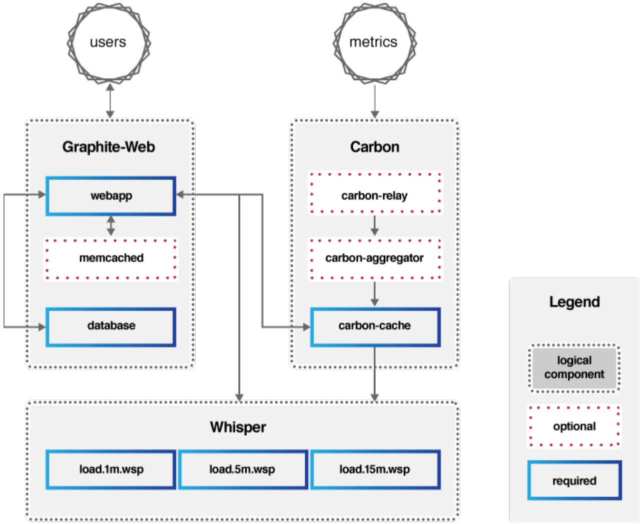


Characteristics

- **Pull-based Metrics Collection:** Prometheus follows a pull-based model where it periodically scrapes metrics data from instrumented targets.
- **Multi-dimensional Data Model:** Prometheus uses a flexible data model with key-value pairs called labels, allowing for multidimensional data collection and querying.
- **Powerful Query Language (PromQL):** Prometheus provides a powerful query language called PromQL for querying and aggregating metrics data, enabling advanced analysis and visualization.
- **Alerting:** Prometheus includes built-in support for alerting based on predefined rules and thresholds, allowing administrators to receive notifications when certain conditions are met.
- **Time Series Database:** Prometheus stores metrics data in a time series database, enabling efficient storage, retrieval, and querying of historical data.
- **Integration with Grafana:** Prometheus integrates seamlessly with Grafana, a popular visualization tool, allowing users to create rich dashboards and graphs based on Prometheus metrics.

Graphite

Source: https://graphiteapp.org/

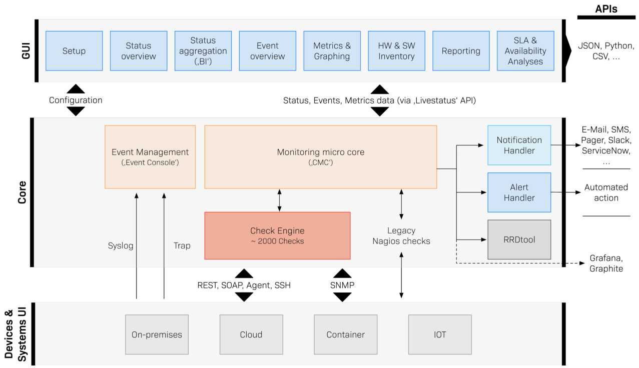


Characteristics

- **Time Series Database:** Graphite is built around a time series database that stores metrics data in a hierarchical structure called a metric namespace. **Carbon Daemon:** Graphite uses a component called Carbon to receive, process, and store metrics data sent by various sources, including applications, servers, and monitoring agents.
- **Whisper Storage Backend:** Graphite uses Whisper as its default storage backend, which is optimized for storing time series data efficiently.
- **Graphing and Visualization:** Graphite provides built-in graphing and visualization capabilities for creating charts, graphs, and dashboards based on metrics data.
- **Integration with Grafana:** Similar to Prometheus, Graphite can be integrated with Grafana for advanced visualization and dashboarding capabilities.
- **Alerting:** Graphite itself does not include built-in alerting capabilities, but it can be integrated with external alerting tools or plugins to add alerting functionality.

Checkmk

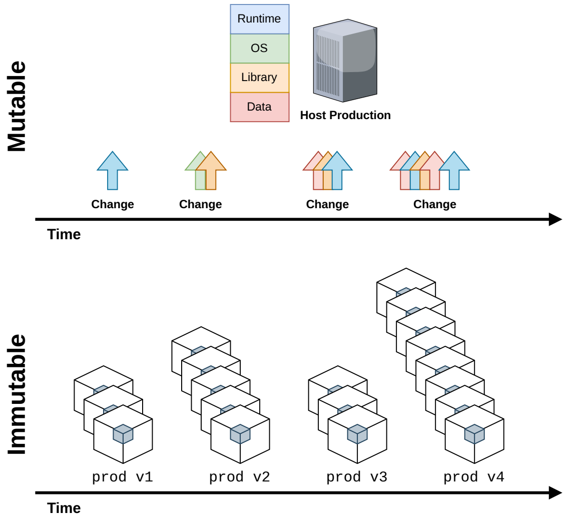
Source: https://docs.checkmk.com/latest/en/cmc.html



Characteristics

- **Agent-based and Agentless Monitoring:** Checkmk supports both agent-based and agentless monitoring methods, allowing administrators to monitor a wide range of devices and platforms.
- **Service Discovery:** Checkmk includes automatic service discovery capabilities, which can dynamically detect and monitor new services or devices added to the network.
- **Rich Set of Plugins:** Checkmk offers a large number of built-in monitoring plugins for monitoring various technologies and applications, as well as the ability to create custom plugins.
- **Event Console:** Checkmk includes an event console for managing and correlating monitoring alerts, events, and notifications in real-time.
- **Role-based Access Control (RBAC):** Checkmk provides RBAC capabilities for managing user access and permissions, allowing administrators to control who can access monitoring data and perform specific actions.
- **Integration with Grafana:** Checkmk can be integrated with Grafana for advanced visualization and dashboarding capabilities, allowing users to create custom dashboards based on monitoring data.

Immutable infrastructures



Characteristics

- **Configuration as Code:** Immutable infrastructures rely on the concept of configuration as code, where infrastructure configurations are defined and managed using code-based configuration files.
- **Automated Deployment:** Immutable infrastructures emphasize automated deployment processes, where infrastructure components are provisioned, configured, and deployed automatically using infrastructure as code (IaC) tools and automation frameworks.
- **Immutable Server Images:** In immutable infrastructures, server or machine images are created with all necessary configurations and dependencies pre-installed.
- **Disposable Infrastructure:** Immutable infrastructures treat infrastructure components as disposable entities that can be created, replaced, and discarded easily.
- **Rolling Updates:** To apply updates or changes to the infrastructure, immutable infrastructures use rolling updates or blue-green deployments, where new instances are deployed alongside existing ones, and traffic is gradually shifted to the new instances.
- **Resilience and Fault Tolerance:** Immutable infrastructures promote resilience and fault tolerance by minimizing the impact of failures and reducing the risk of configuration drift.

Managing Infrastructures – Infrastructure as Code (IaC)

Infrastructure as Code (IaC) is a method in system administration where infrastructure configurations and provisioning are managed and automated through code rather than manual processes. With IaC, infrastructure components such as servers, networking, and storage resources are defined, configured, and managed using code-based configuration files, scripts, or templates.

Benefis

- **Consistency:** IaC ensures that infrastructure configurations are consistent across environments, such as development, testing, staging, and production.
- **Automation:** IaC enables automation of infrastructure provisioning and management tasks, reducing the need for manual intervention and human error.
- **Scalability:** With IaC, organizations can easily scale their infrastructure up or down in response to changing demand.
- **Version Control and Auditability:** Infrastructure code is treated like application code and stored in version control systems such as Git.
- **Reusability and Modularity:** IaC promotes reusability and modularity by allowing common configurations to be abstracted into reusable components or modules.

Source of examples

The examples were taken from the book of Daniel Stender *„Cloud-Infrastrukturen: Das Handbuch für DevOps-Teams und Administratoren“*, Rheinwerk Computing (2020), ISBN 978-3-8362-6949-0.

Vagrant (1/2)

Vagrant is an open-source[1] tool for building and managing portable development environments. It allows developers to create reproducible, shareable, and lightweight virtualized environments that closely resemble production setups.

Characteristics

- **Portable Development Environments:** allows developers to create and manage reproducible development environments that are easily portable across different machines and platforms.
- **Configuration Management:** simplifies configuration management by allowing developers to define environment settings, provisioning scripts, and dependencies using simple configuration files.
- **Integration with Virtualization Providers:** supports various virtualization providers, including VirtualBox, VMware, and Docker, allowing developers to choose the most suitable environment for their projects.

[1] From August 10, 2023 on the licensing has changed to Business Source License (BUSL)!

Vagrant – simple example (2/2)

```ruby
Vagrant.configure("2") do |config|
  config.vm.box = "bento/ubuntu-22.04"
  # Example for VirtualBox:
  config.vm.provider "virtualbox" do |vb|
  # Configure the network of the machine
  config.vm.network "private_network", ip: "192.168.56.10"
  # Display the VirtualBox GUI when booting the machine
  vb.gui = true
  # Customize the amount of memory on the VM:
  vb.memory = "1024"
  end
  # Enable provisioning with a shell script. Additional provisioners such as Ansible
  config.vm.provision "shell", inline: <<-SHELL
  apt-get update
  apt-get install -y apache2
  SHELL
end
```

- The example Vagrantfile presents the installation of a Ubuntu instance.
- The script also configures the network and installs an Apache webserver.
- The example could be enhanced with more complex provision scenarios and combined with other tools e.g. Ansible.

Simple setup

Only the follwing commands are necessary:

- `vagrant up`
- `vagrant ssh`
- `vagrant destroy`

Terraform (1/2)

Terraform is an open-source[1] infrastructure as code tool used for building, changing, and versioning infrastructure safely and efficiently. It allows users to define infrastructure configurations using declarative code in a simple, human-readable language.

There is the alternative **OpenTofu**, which is a fork of Terraform that is open-source, and managed by the Linux Foundation!!! Link:`https://opentofu.org/`

Characteristics

- **Multi-Cloud Support:** supports multiple cloud providers (such as AWS, Azure, Google Cloud Platform) and infrastructure services, as well as on-premises environments.
- **State Management:** maintains a state file that tracks the current state of the infrastructure managed by Terraform. This state file helps Terraform understand the relationships between resources and enables it to plan and execute changes to the infrastructure in a safe and predictable manner.
- **Modularity and Reusability:** allows users to define infrastructure configurations using modular and reusable components called modules.

[1] From August 10, 2023 on the licensing has changed to Business Source License (BUSL)!

Terraform – simple example (2/2)

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.16"
    }
  }
  required_version = ">= 1.2.0"
}
provider "aws" {
  region  = "us-west-2"
}
resource "aws_instance" "app_server" {
  ami           = "ami-{id}"
  instance_type = "t2.micro"
  tags = {
    Name = "ExampleAppServerInstance"
  }
}
```

- The provider is used to create and manage resources (here `aws`).
- Multiple provider blocks can be used to manage resources from different providers.
- The resource blocks define components of the infrastructure (e.g. physical or virtual component, here an EC2 instance).

Simple setup

Only the follwing commands are necessary:

- `terraform init`
- `terraform apply`
- `terraform show`
- `terraform apply -destroy`

Ansible (1/3)

Ansible is an open-source automation tool used for configuration management, application deployment, and task automation. It allows users to define infrastructure configurations and automate complex tasks using simple, human-readable YAML-based playbooks.

Characteristics

- **Agentless Architecture:** operates over SSH, eliminating the need for agent installation on managed nodes.
- **Declarative Configuration Management:** uses a simple, human-readable YAML-based language to define infrastructure configurations and automation tasks.
- **Orchestration and Automation:** provides powerful orchestration capabilities for automating complex workflows and tasks across multiple servers and environments.
- **Idempotent Operations:** ensures idempotent operations, meaning that applying the same configuration multiple times results in the same outcome.

Ansible concepts (2/3)

Ansible has the following basic concepts:

Host: A remote machine managed by Ansible.

Group: Several hosts grouped together that share a common attribute.

Inventory: A collection of all the hosts and groups that Ansible manages.

Modules: Units of code that Ansible sends to the remote nodes for execution.

Tasks: Units of action that combine a module and its arguments along with some other parameters.

Playbooks: An ordered list of tasks along with its necessary parameters that define a recipe to configure a system.

Roles: Redistributable units of organization that allow users to share automation code easier.

YAML files: A popular and simple data format that is very clean and understandable by humans.

Ansible – example (3/3)

Vagrantfile

```ruby
Vagrant.configure(2) do |config|

  config.vm.box = "bento/ubuntu-22.04"

  config.vm.provision "ansible" do |ansible|
    ansible.verbose = "v"
    ansible.playbook = "playbook.yml"
  end
end
```

Playbook

```yaml
---
- name: Install Nginx
  hosts: all
  become: yes

  tasks:
    - name: Update apt package cache (Debian/Ubuntu)
      apt:
        update_cache: yes
      when: ansible_os_family == 'Debian'

    - name: Install Nginx (Debian/Ubuntu)
      apt:
        name: nginx
        state: present
      when: ansible_os_family == 'Debian'

    - name: Start Nginx service (Debian/Ubuntu)
      service:
        name: nginx
        state: started
      when: ansible_os_family == 'Debian'
```

- The listings setup a VagrantBox inside of VirtualBox and provision the machine with Ansible.
- The Playbook installs `nginx` as an alternative to Apache.

Packer

Packer is an open-source[1] tool for automating the creation of machine images for multiple platforms, such as AWS, Azure, VMware, and Docker. It allows users to define machine configurations using declarative JSON templates, enabling consistent and reproducible image builds across different environments.

Characteristics

- **Multi-Platform Support:** Packer supports multiple platforms and providers, including AWS, Azure, Google Cloud Platform, VMware, VirtualBox, and Docker.
- **Provisioning:** Packer supports various provisioning methods, including shell scripts, Ansible playbooks, Chef recipes, and Puppet manifests. Automates the installation and configuration of software packages, dependencies, and applications within the machine image.
- **Parallel Builds:** Packer can perform parallel builds to speed up the image creation process, especially when building images for multiple platforms or configurations simultaneously.
- **Idempotent Operations:** produces the same result whether executed once or multiple times with identical inputs.

[1] From August 10, 2023 on the licensing has changed to Business Source License (BUSL)

Cloud-Init

Cloud-init is an open-source package used for initializing cloud instances when they first boot up. It allows users to customize and configure cloud instances during the initial provisioning process using simple configuration files written in YAML or JSON format. Cloud-init supports a wide range of cloud platforms, including AWS, Azure, Google Cloud Platform, and OpenStack.

Characteristics

- **Initialization and Configuration:** a package uses a package for initializing cloud instances when they first boot up. It allows users to customize cloud instances during the initial provisioning process using simple configuration files written in YAML or JSON format.
- **Multi-Cloud Support:** supports a wide range of cloud platforms, including AWS, Azure, Google Cloud Platform, and OpenStack, as well as some virtualization platforms like VMware and VirtualBox.
- **User Data:** processes user data provided during instance creation, allowing users to pass custom configuration scripts, user accounts, SSH keys, and other initialization tasks to cloud instances.
- **Integration with Cloud Metadata Services:** retrieves instance metadata and user data from cloud metadata services provided by cloud platforms.

Cost of on-premise operation

The traditional operation of IT infrastructures produce high costs and efforts, which need to be calculated by companies!

The costs of traditional IT infrastructures usually stem from the following facts:

- Purchase costs of hardware (e.g. servers, networking, general equipment).
- Housing for the infrastructure (e.g. server enclosure, etc.).
- Operational costs for the infrastructure (e.g. power, maintenance, etc.).
- Cost for personnel installing, configuring an maintaining the infrastructure.
- Additional costs for updating and upgrading of infrastructure components (e.g. servers, networking, etc.)

Case Study for example calculation (1/3)

Case Study

As an example we can consider a small to medium enterprise (SME), going into business. We will make assumptions on the demands of that company in order to have a common ground for the calculation of the expected costs for the operation of the service offering.

Source of example calculation

K. Konstantinos, M. Persefoni, F. Evangelia, M. Christos, and N. Mara, 'Cloud computing and economic growth', in Proceedings of the 19th Panhellenic Conference on Informatics, Athens, Greece, 2015, pp. 209–214.

Case Study for example calculation (2/3)

Example company

The example company is a news company streaming news and has 50 employees. For the operation and offering of the news service the company needs an IT infrastructure, consisting of networking, web hosting, backups, etc.

Assumptions

The software and services used for the installation of the service are open source and therefore free of charge. Maintenance costs are the same for the calculation of the traditional and the cloud service scenario. The estimations are based on a duration of three years, which is a realistic time frame!

Case Study for example calculation (3/3)

Hardware specification for the service on-premise

5 Servers equipped with Intel Xeon E5-2640 v2 (8 core, 2 GHz, 20MB, 95W) CPUs.

Each one having the following characteristics:

Processors	2 x 8 cores per processor
RAM	16GB of RAM memory
Networking	4 NICs, 4 ports per controller
Storage	5TB SAS
Size	1 Unit
Power Supply	460W

First we want to calculate the **total cost of ownership (TCO)** for an on-premise operation of the service in a traditional IT infrastructure. Therefore we need to calculate the **capital (CapEx) and operational expenditures (OpEx)**.

Traditional IT-Infrastructures (1/3)

CapEx

CapEx are large investments in fixed assets over a longer period.

- IT infrastructure (servers, networking, software, etc.)
- IT equipment
- Data center housing
- Infrastructure maintenance

OpEx

OpEx are costs associated with day-to-day operations.

- Business-related operating costs (on-demand rent, utilities, salaries, etc.)
- Cloud-based software or service subscription fees (SaaS, PaaS, IaaS, etc.)
- Software and service support
- Data center or off-premises cloud costs

Traditional IT-Infrastructures - CapEx (2/3)

Initial Cost of Infrastructure	**Quantity**	**Cost**
Servers	5	17.500 €
Total Storage (SAN)	5TB	35.000 €
Networking (Switch)	4	14.710 €
Faclities (PDU,KVM etc.) per rack	1	897,00 €
Cooling equipment per rack	1	717,00 €
Capital Expenditures		**68.824 €**

Traditional IT-Infrastructures - OpEx (3/3)

OpEx (3 years)		Price	Annual Cost	Three Year Cost
Actual Operating Power[2]	308 Watts per server	0,22€/kwh	2.962 €	8.885 €
Actual Cooling Power[3]	385 Watts per server	0,22€/kwh	3.702 €	11.106 €
Real Estate Rent	5 sq.m	5€/sq.m	300 €	900 €
Operating Expenditures			**6.964 €**	**20.891 €**

Total cost of ownership (TCO)

This calculation demonstrates a total cost of ownership (TCO) of **89.715 €** over three years!

[2]With current prices (approx. 0,38€/kwh) 5126 € resp. 15.379 €!

[3]With current prices (approx. 0,38€/kwh) 6407 € resp. 19.223 €!

Example calculation of costs using AWS

Table: Cost of one VM instance in AWS

AWS VM (24/7)	Cost per month	One Year	Three Years
EC2 m2.xlarge + 1 TB SSD EBS	145 €	1740 €	5.398 €
Transfer	75 €	900 €	2.707 €
Load balancer	22 €	264 €	780 €
Cloud Object Storage capacity	28 €	336 €	991 €
Cloud Object Storage requests	48 €	576 €	1.743 €
Total	**323 €**	**3878 €**	**11.619 €**

Traditional IT vs AWS

Table: Comparison between traditional IT and AWS

Period	Traditional IT	AWS
Start	68.824 €	0 €
1st Year	6.964 €	19.364 €
2nd Year	6.964 €	19.364 €
3rd Year	6.964 €	19.364 €
Total	**89.715 €**	**58.093 €**

Cost savings of example

This calculation demonstrates possible cost savings of **more than 35%** over three years!

Traditional IT vs AWS

More points to ponder

The hosting of the service in the cloud has also the following additional benefits:

- The provider is responsible for the operation of hardware!
- The provider is responsible for the availability of the service!
- The provider is responsible for upgrades of the infrastructure!
- Additional demands can be fulfilled in an elastic manner!

Should we get rid of traditional IT?

Answer. . . **NO!**

Some amount of IT is still needed! Also depending on the use case of the operation scenario on-premise operation is still crucial! We also have to consider the dangers of a vendor-lock-in! More on this in **slide set 4**!

Summary

In this lecture we have discussed the following topics:

- Legacy IT, server technology and the relation ship to cloud computing
- The benefits of server virtualization and consolidation with virtual machines
- The foundations of Cloud Computing and the technologies enabling it
- The different types of virtual networking and storage technologies
- The difference between hardware virtualization and operating system virtualization
- Managing infrastructures through code definitions
- The price of traditional IT and the beneficial effects of Cloud Computing on costs

Outlook

~~1st part:~~	~~Introduction~~
2nd part:	Technological foundations $\Longleftarrow$ *This slide set*
3rd part:	Service models, deployment models
4th part:	Adoption and strategy
5th part:	Architectures and applications
6th part:	Cloud-Native applications
7th part:	Current and future trends

3rd part: Service models, deployment models

Topics:

- Deployment models in Cloud Computing
- Service models in Cloud Computing
- Public Cloud Computing offerings
- Private Cloud Computing offerings

Thank You For Your Attention!

Henry-Norbert Cocos, M.Sc
Frankfurt University of Applied Sciences
Room 1-230
☎ +49 69 1533-2699
✉ `cocos@fb2.fra-uas.de`
`www.henrycocos.de`