CS5224 Cloud Computing

Principles of Cloud Computing

Introduction

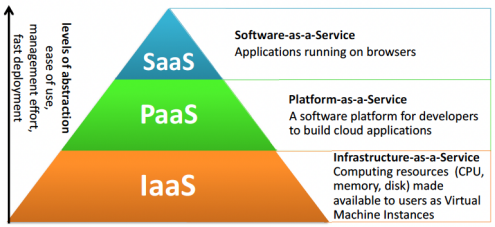
- Three eras of computing:
  1. **Tabulating** (1890s - 1940s): mechanical tabulators, limited to a *single task*.
  2. **Programming** (1950s - 2010s): programmable systems to perform **multiple different tasks**.
  3. **Cognitive** (2011-): systems which **learn and adapt over time**, extending the boundaries of human cognition.
     - Enables discovery, engagement, and decision.
- **Cloud computing**: platform for *hosting & delivering* cognitive services.
  - **5 key attributes/characteristics**:
    1. On-demand self service: assists with **capacity planning**, enabling high **availability**.
       - Challenges of capacity planning: usage/demand fluctuations, cost of resource provisioning, etc.
         - **Provisioning for peak load**: difficulty anticipating peak load, resources wasted during non-peak times.
         - **Under-provisioning**: loss of potential revenue from users not served.
       - Strategies:
         - **Lead strategy**: add capacity in anticipation of demand.
         - **Lag strategy**: add capacity when resources reach their full capacity.
         - **Match strategy**: add capacity in small increments as demand increases.
         - **Cloud**: automatic resource provisioning.
    2. Broad network access (ubiquitous access).
    3. Location-independent resource pooling (i.e., **multi-tenancy**).
       - **Single-tenant**: each cloud consumer has a separate IT resource instance.
       - **Multi-tenancy**:
         - A cloud provider pools its IT resources to serve multiple cloud service consumers.
         - An instance of the program serves different consumers, with each instance isolated from the other.
         - Reduces costs (e.g., for maintenance) for the cloud provider.
    4. Rapid **elasticity**: enables fast deployment + organizational agility.
       - For on-premise systems, upfront investments and infrastructure ownership costs may be prohibitive.
       - Cloud computing enables **better organizational agility** due to resource elasticity.
    5. Measured service: typically **reduces business costs**.
       - On-premise systems have several costs associated with them (e.g., upfront investment/ownership costs, operational costs, etc.).
         - Cloud computing offers **cost efficiency at scale**.
       - Compared to **scaling-out**: lower opportunity cost (i.e., excess capacity).
         - **Horizontal scaling**: add the **same resource type**.
       - Compared to **scaling-up**: lower capital expenditure.
         - **Vertical scaling**: replace the resource with **higher/lower capacity**, or add **resources to a single node**.

	Horizontal	Vertical
Cost	Less expensive, using commodity hardware.	More expensive, using specialized hardware.
Availability	Resources instantly available.	Resources normally instantly available.
Ease of setup	Resource replication and automated scaling.	May need additional setup.
Resources	Additional IT resources needed.	No additional IT resources needed.
Hardware capacity	Not limited by hardware capacity.	Limited by maximum hardware capacity.

  - Other concepts:
    1. **Cloud-based IT resources**: include
       - Physical servers.
       - Virtual servers.
       - Software programs.
       - Cloud services.
       - Storage devices.
       - Network devices.
    2. **Cloud service**: any IT resource made *remotely accessible* via cloud.
       - Cloud service **consumer**: role assumed by a software program that accesses a cloud service.
       - **Service usage models**: SaaS, PaaS, and IaaS.
  - Challenges:
    - Technical:
      1. Software development: different cloud platforms and services across cloud providers.
      2. Continuously evolving tools.
      3. High costs of moving large data.
      4. Security.
      5. Internet dependence.
      6. Quality of service depends on the provider.
      7. High energy consumption.
    - Non-technical:
      1. **Increased security vulnerabilities**: responsibility over data security becomes shared with cloud providers.
         - **Expansion of trust boundary** → introduce new vulnerabilities.
           (a) **Trust boundary**: the demarcation boundary that separates the components and systems that are considered trustworthy from those that are not.
         - **Overlapping trust boundaries** due to shared IT resources across different cloud consumers → introduce opportunities for malicious cloud consumers to steal/damage business data.
      2. **Reduced operational governance control**: different levels of IT resource governance between on-premise and cloud services.
         - Unreliable cloud providers may not maintain/meet SLA guarantees.
         - Unreliable network connections may compromise the quality of communication between cloud consumer and cloud providers.
      3. **Privacy/legal issues**: data centers are typically set up in affordable/convenient geographical locations.
         - Problems with **data localization/residency**: there may be industry/government regulations on data privacy or storage policies, e.g., US-TTP.
         - Legal issues: laws may require data to be disclosed to the government agency → tension between personal rights (privacy) and society.
      4. Vendor lock-in.
      5. Service level agreements.

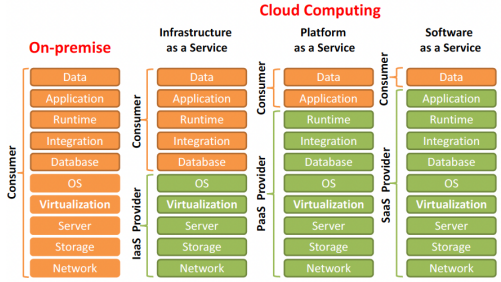
Concepts and models

- **Cloud service models**: different models for delivering cloud services.
  1. **Software-as-a-Service (SaaS)**: consumers use providers' applications by providing data to obtain the results.
     - Consumers access a cloud service without knowledge of the underlying IT resources/implementation details.
     - E.g., Salesforce, Google apps.
  2. **Platform-as-a-Service (PaaS)**: consists of 2 main parts, *platform software* and the *computing resources* needed to run/host the platform software.
     - Consumers deploy customer-related applications to the cloud.
     - Consumers do not need to know the implementation details of the platform.
     - E.g., AWS, Azure, Google cloud.
  3. **Infrastructure-as-a-Service (IaaS)**: lowest level of abstraction, physical resources are abstracted as virtual resources.
     - Consumers rent processing, storage, network capacity, and other fundamental computing resources.
     - Consumers are provided with a range of contractual guarantees by the cloud provider, e.g., on *capacity*, *performance*, and *availability*.
     - E.g., virtual servers, Amazon EC2 object storage.
  4. **Function-as-a-Service (FaaS)**: serverless computing.

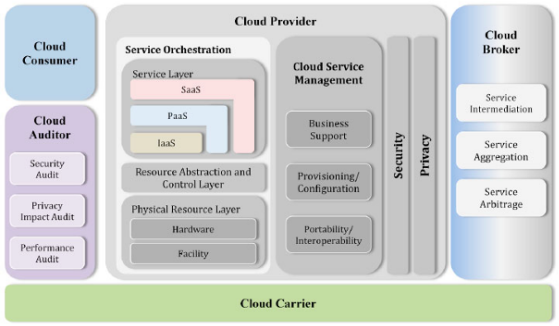


  - Some differences between the different service models include:

Service Models	Pros	Cons
on-premise	highest flexibility	long time-to-market
IaaS	scalability, no hw procure	privacy
PaaS	DB, Frameworks, middleware ready	vendor lock in
SaaS	time-to-market	lowest flexibility



- **Cloud reference architecture**: focuses on *what* cloud services provide, and *how* the 5 key actors interact in a cloud.
  - **NIST reference model**: provides a common reference framework to describe, discuss, and develop a system-specific architecture.



  - Major actors:
    1. **Consumer**: maintains business relationships with and uses services from cloud providers.
       - 3 types: SaaS consumer, PaaS consumer, and IaaS consumer.
    2. **Provider**: offers cloud service(s) to cloud consumers.
       - **SaaS provider**: develops, deploys, configures, maintains, and updates operation of the software applications on a cloud infrastructure to provision services at the expected SLAs to cloud consumers.
       - **PaaS provider**: develops and manages computing infrastructure for the platform, and runs cloud platform component software (e.g., runtime software execution stack, databases).
       - **IaaS provider**: provides virtual resources by acquiring physical computing resources underlying the service, including servers, networks, storage, and hosting infrastructures.

       Main responsibilities of providers:
       (a) **Service deployment**: whether cloud infrastructure is operated as a public, private, community, or hybrid cloud service.
       (b) **Service orchestration**: provisioning/composing cloud services to be delivered to cloud consumers. Typically consists of 3 layers:
         i. **Service layer**: defines interfaces for cloud consumers to access the computing resources.
         ii. **Resource abstraction** and **control layer**: software abstraction of system components that cloud providers use to provide and manage access to the physical computing resources.
         iii. **Physical resource layer**: includes all physical computing resources.
       (c) **Service management**: service-related functions that are required to manage and operate the cloud services offered to cloud consumers, e.g.:
         i. **Business support**: set of business-related services to deal with clients and supporting processes e.g., accounting, billing, pricing, and rating.
         ii. **Provisioning/Configuration**: includes automatic resource deployments, SLA managements, etc.
         iii. **Portability**: whether cloud consumers can move their data or applications across multiple cloud environments at low cost and with minimum disruption.
         iv. **Service interoperability**: whether cloud consumers can use their data and services across multiple cloud providers with a unified management interface.
         v. **System portability**: whether cloud consumers can migrate a virtual-machine instance or a machine image from one provider to another, or migrate applications & services and its contents from one provider to another.
       (d) **Security**: consumers are exposed to different entry points into cloud systems, introducing different attacking surfaces for adversaries.
         - Providers need to address security requirements e.g., **authentication**, **authorization**, **availability**, **confidentiality**, identity management, integrity, audit, security monitoring, incident response, and security policy management.
       (e) **Privacy**: providers should ensure proper and consistent collection, processing, communication, use, and disposing of personal information and personally identifiable information in the cloud.
    3. **Auditor**: conducts independent assessments of cloud services, system operations, performance, and security of the cloud implementation.
       - Verifies compliance with regulation and security policies, and conformance to standards through review of objective evidence.
    4. **Broker**: manages the **use**, **performance**, and **delivery** of cloud services, and negotiates relationships between cloud providers and cloud consumers. Main services include:
       (a) **Service intermediation**: enhances a given service by providing value-added services to cloud consumers, such as managing access to cloud services, identity management, performance reporting, enhanced security, etc.
       (b) **Service aggregation**: combines and integrates multiple services into one or more new services, provides data integration, and ensures the secure data movement between the cloud consumers and providers.
       (c) **Service arbitrage**: flexibility to choose services from multiple agencies; similar to service aggregation except that the service being aggregated are not fixed.
    5. **Carrier**: provides **connectivity and transport of cloud services** from cloud providers to consumers.
       - Providers set up SLAs with a cloud carrier to provide services consistent with the level of SLAs offered to cloud consumers.
- **Cloud deployment models**: how cloud services are exposed/deployed for use.
  - **Organizational boundary**: physical scope of IT resources owned and governed by an organization.
  - **(Logical) trust boundary**: logical perimeter representing which IT resources are trusted by an organization.
  1. **Private cloud**: used solely by an organization, for enterprises/corporations with large scale IT.
     (a) **On-site**: cloud service consumer in an organization's on-premise environment accesses a cloud service hosted on the same organization's private cloud via a VPN.
     (b) **Outsourced**.
  2. **Public cloud**: shared by all consumers, open market for on-demand computing and IT resources.
  3. **Community cloud**: shared by several organizations, supporting a specific community.
  4. **Hybrid cloud**: an inter-operation of public/private clouds, extends private cloud(s) to include a shared public cloud.
  5. **Sovereign cloud**: stores data (incl. metadata) within national borders, adhering to local data protection laws.
     - To meet regulatory and compliance needs of a country/jurisdiction.

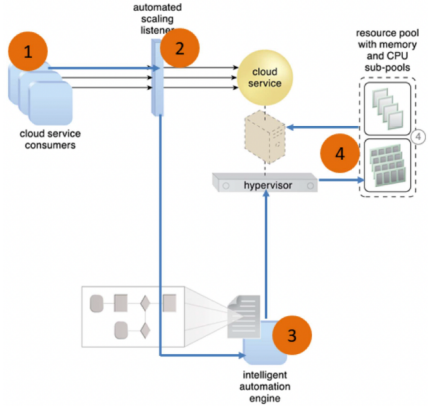
Cloud architecture

- **Rapid elasticity**: cloud providers have to respond to fluctuations in:
  1. **Cloud consumers' resource demands**, in two main dimensions:
     (a) **# jobs/txs** that are running.
     (b) **Resource demand within a job execution**.
  2. **Available cloud resources**, due to consumers' fluctuating resource demands.
- Providers' main objectives in organizing cloud resources:
  1. **Achieve elasticity/scaling**: consumers must be able to add/return cloud resources depending on their resource demands.
  2. **Achieve balanced utilization of cloud resources** through workload distribution, to minimize over- or under-utilization of resources.
- **How to organize/partition resources**:
  1. **Workload distribution**: *load balancer* distributes workload to reduce over- or under-utilization of resources (e.g., Amazon elastic load balancing (ELB) service). Additional mechanisms include:
     - **Cloud usage monitor**: tracks consumer resource demands.
     - **Audit monitor**: checks that consumers adhere to resource pool usage policies/agreements.

     Typically, consumer requests for cloud services are directed to a load balancer, which matches consumer resource demands with available cloud resources, and distributes the workload evenly across the provider's resource pool.
  2. **Resource pooling**: aggregates different types of cloud resources to serve different consumer needs. Examples include:
     - **Dedicated pools**: each type of resource is grouped into a sub-pool.
     - **Complex pools**: hierarchical structure with parent, sibling, and nested pools.
       - **Parent resource pool**: typically consists of a collection of sub-pools which is a superset of its children.
       - **Sibling resource pool**: physically grouped but logically isolated, so that each consumer has access to an independent pool.

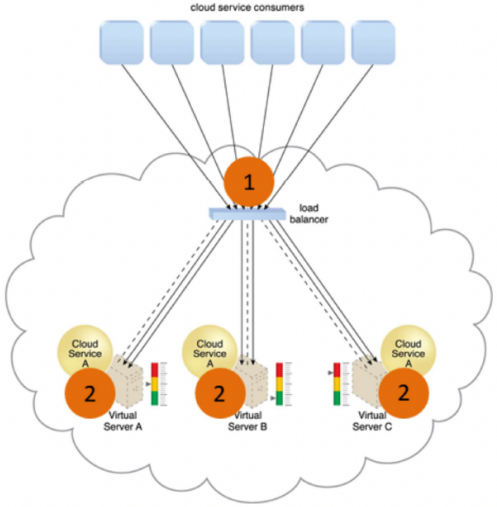
    - **Nested resource pool**: smaller pools with the *same type of resources*, typically for assigning to different departments/groups in the same organization.

  Additional mechanisms include:
  - **Pay-per-use monitor**: monitoring of resource usage so that consumers are billed for resources used.
  - **Resource management & replication**: providers consumers with tools for administering and generating new instances of resources from the resource pools.
3. **Dynamic scalability**: enables variable resource utilization to meet *usage demand* fluctuations; dynamic resource allocation is based on predefined *scaling conditions*.
   (a) **Dynamic horizontal scaling** (same resource type): scaling of resource instances, e.g., resource replication.
   (b) **Dynamic vertical scaling** (change size): scaling of the processing capacity of a single IT resource, e.g., increase in memory of resource.
   (c) **Dynamic relocation** (different resource types): resource relocated to a host with larger capacity, e.g., relocation of data from tape-based storage to disk-based storage.
4. **Elastic resource capacity**: scripts capturing the administration workflow logic are executed by an *intelligent automation engine* that sends scaling requests to the hypervisor/virtual infrastructure manager (VIM) $\rightarrow$ resolves fluctuations in *resource availability*.

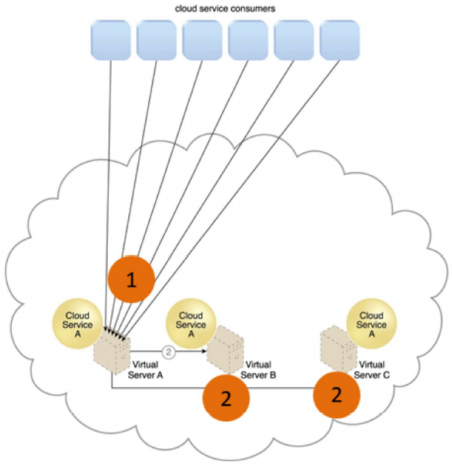


   (a) Cloud service consumers send requests to a cloud service monitored by an **automated scaling listener**.
   (b) An **intelligent automaton engine** deployed with the workflow logic notifies the resource pool using allocation requests.
   (c) When consumer requests increase, the automated scaling listener signals the intelligent automation engine to execute the **workflow logic scripts**.
   (d) The script then signals the hypervisor to allocate more resources from the resource pools.
   (e) The hypervisor allocates additional resources to the virtual service to handle the increased workload.
5. **Service load balancing**: variation of the workload distribution architecture for scaling cloud service implementations (e.g., Amazon auto scaling service).
   - In the workload distribution architecture, jobs are distributed across machines.
   - In the service load balancing architecture, the cloud service is duplicated to run on more than one VM, and consumer demands are distributed across virtual servers.

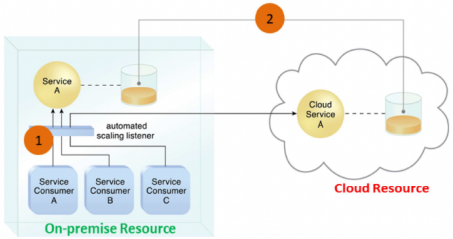
   (a) **Independent load balancer**:
      i. Cloud service is replicated and runs on different virtual servers.
      ii. Load balancer intercepts messages sent by cloud service consumers, forwarding them to the virtual servers for workload processing.



   (b) **Built-in load balancer**:
      i. Cloud service consumer requests are sent to cloud service A on virtual server A.
      ii. Cloud service implementation contains *built-in load balancing logic* capable of distributing requests to the neighbouring cloud service A implementations on virtual servers B and C.



- **How to operate/manage resources to meet consumer objectives**:
  1. **Cloud bursting**: allows users to tap on additional resources on the cloud when they have run out of on-premise resources (i.e., **burst out**).
     - When resource demand decreases, the additional cloud resource can be returned (i.e., **burst in**).
     - Consumer must pre-deploy a redundant copy of the applications/services on the cloud, which is synchronized with the on-premise version, but remains inactive until it is needed.



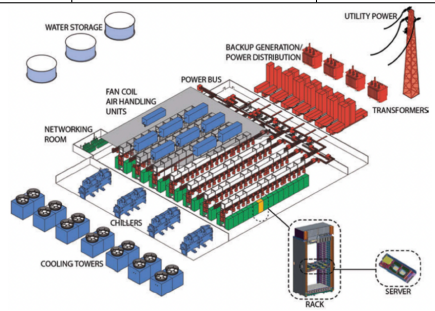
     (a) An **automated scaling listener** monitors the usage of the on-premise service A.
     (b) When the usage threshold is exceeded, consumer requests to service A are redirected to A's redundant implementation in the cloud.
     (c) A **resource replication system** ensures that the state management databases are synchronized.
  2. **Elastic disk provisioning**: achieves dynamic storage provisioning and charging based on actual usage (instead of being based on fixed disk storage allocation).
     - Requires runtime monitoring of disk usage, which can be incorporated as part of the virtual server software.

Cloud Computing Technology

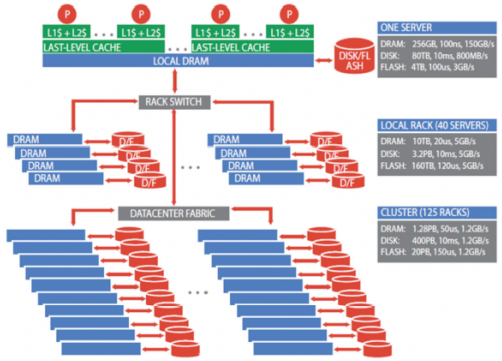
Resource hosting & data centers

- **Resource hosting**: how are cloud consumers and providers connected?

	On-premise	Cloud-based
Quality of service (QoS)	Complete control over QoS.	Dependent on using multiple ISPs.
Security	Safeguarded using firewalls and monitoring software.	Dependent on ISPs.
Network latency & bandwidth	Lower latency, higher bandwidth.	Higher and more variable latency.



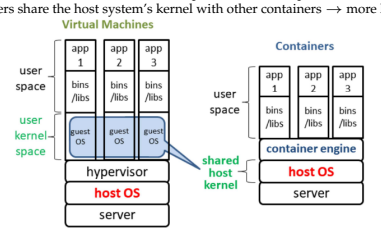
- **Key components** of a data center:
  1. **Physical IT resources**: main server hall includes servers, storage, and network.
     - **Machine rack**: supports servers, storage, and networking equipment by enabling power conversion & delivery, and providing battery backup.
     - **Server**: consists of multiple processors, each with many cores, and a hierarchy of memory and interfaces for hard disks.
       - 1 cluster $\leftrightarrow$ multiple servers.
     - **GPUs**: located in a *PCIe-attached accelerator tray* containing multiple GPUs.
       - Within a tray, GPUs use interconnects such as *NVlink*.
       - Multiple GPU trays are connected to the datacenter with NICs, communicating directly through PCIe.
     - **Storage**: there are 2 categories of data
       (a) **Private/individual running tasks**: reside in local DRAM or disk.
       (b) **Shared, distributed workload**: distributed across the storage network.
       As we move from one server to all racks in a datacenter, storage capacity increases but network latency increases as well.



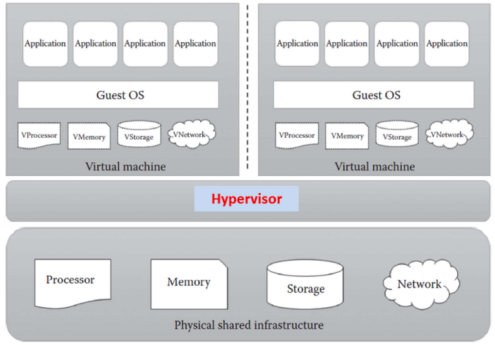
     - **Network**:
       - Between data center (LAN) and cloud consumers (WAN): **carrier interconnection**.
       - Within data centers: **LAN** connects servers, **storage area network (SAN)** connects servers and storage systems.
  2. Supporting resources:
     - **Building**: i.e., data center.
     - **Power**: an uninterruptible power supply (UPS) is needed to keep the data center alive during power outages. This consists of:
       - **Battery power backup**: keep system alive when power supply disruption is detected.
       - **Power generator**: supply power for a longer duration before the outage is resolved.
     - **Cooling**: may exist in the form of air cooling or water cooling.
       - Water cooling is more efficient than air cooling, but the equipment is more expensive.
  3. Safety: **fire protection** and **security** (physical + cybersecurity).
- **Data center tiers**: based on degrees of availability (i.e., downtime per year).
  1. **Tier I** (99.671% uptime, 29h/yr downtime): single path for power distribution, no redundant components.
  2. **Tier II** (99.741% uptime, 23h/yr downtime): adds redundant components.
  3. **Tier III** (99.982% uptime, 1.5h/yr downtime): one active and one alternate distribution path for utilities, each path contains redundant components (ensures concurrent maintainability).
  4. **Tier IV** (99.995% uptime, 0.5h/yr downtime): two simultaneous active power and cooling distribution paths, with redundant components on each path.
- **Energy usage** and **power efficiency**: data centers comsume a lot of energy.
  - **Power usage efficiency (PUE)**: PUE $= \frac{\text{total power used by data center}}{\text{power used by IT equipment}}$, ideal PUE $= 1$.
  - **Energy-proportional systems**: energy consumed is proportional to the amount of work done.
    - Even when power requirements scale linearly with the load, energy efficiency is not a linear function of load (e.g., idle systems may use 50% of power).
    - An energy-proportional system consumes no power when idle, very little power under a light load, and gradually more power as the load increases.
    - **Dynamic power range**: measure of energy proportionality, by taking the low and upper range of the power consumed, and expressing it as a percentage.
      - Wider range $\rightarrow$ higher/better energy proportionality.

Virtualization & multi-tenancy

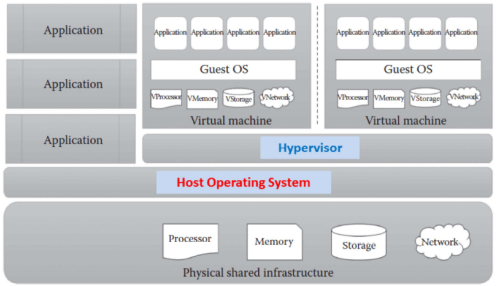
- **Virtualization**: allows multiple OS to run and share a single hardware $\rightarrow \uparrow$ resource utilization, $\downarrow$ cost.
- $(+)$ **Hardware independence**: eliminates software-hardware dependencies.
- $(+)$ **Resource consolidation**: different virtual resources share one physical resource.
- $(+)$ **Resource replication**: virtual resources are created as **virtual disk images**, which are accessible to the host's OS, and enable agility in migration, deployment, and recovery (through snapshots).
- $(-)$ **Performance overhead**.
- $(-)$ **Single point of failure** for these virtualized resources.
- **Containers** vs **virtual machines (VMs)**:
  - Containers and VMs both allow the existence of multiple isolated user space instances.
  - Unlike VMs, containers share the host system's kernel with other containers $\rightarrow$ more lightweight.



- **Hypervisor/virtual machine manager (VMM)**: sits between VMs and physical infrastructure, provides the required virtual infrastructure for VMs.
  - **Type 1**: bare metal/native, runs on physical infrastructure without any help from the host OS.



  - **Type 2**: hosted/embedded, requires the help of host OS to communicate with the underlying infrastructure.



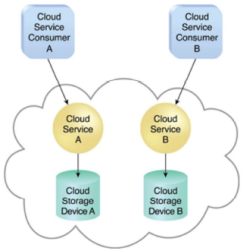
- **Types** of virtualization:
  1. **Processor virtualization**: abstracting a physical processor to a pool of virtual processors.
  2. **Memory virtualization**: providing virtual main memory to VMs that are abstracted from the physical main memory.
  3. **Storage virtualization**: abstracting multiple (shared) physical storage as a pool of logical storage.
  4. **Network virtualization**: abstracting physical network components to form a virtual network.
  5. **Data virtualization**: aggregating heterogeneous data from different sources to a single logical/virtual volume of data.
  6. **Application virtualization**: allowing users to access the virtual instance of a centrally hosted application without installation.
- **Approaches** to virtualization:
  1. **Full**: VMs can run different guest OSes, which are abstracted from the hardware.
     - Guest OS communicates with the hardware through the VMM (at ring 0) using binary translation.
     - Guest OS is not aware that it is virtualized.

     (+) Better isolation and security.
     (+) Different OSes can be run simultaneously.
     (+) Easy to install and use.
     (+) Virtual guest OS can be easily migrated to native hardware.
     (−) Binary translation overhead → ↓ performance.
     (−) Need for correct combination of hardware and software.
  2. **Para**: modified guest OS replaces OS requests with *hypercalls*.
     - **Hypercalls**: system calls with privilege to communicate directly between OS and hypervisor without translation.
     - Guest OS is aware that it is running in a virtualized environment.

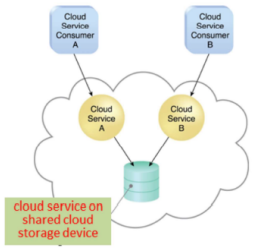
     (+) No overhead of binary translation.
     (+) No need for special hardware.
     (−) Overhead of guest OS kernel modifications.
     (−) Modified guest OS cannot migrate to run on physical hardware.
     (−) VMs lack backward compatibility, difficult to migrate to other hosts.
  3. **Hardware-assisted**:
     - **Hardware/processor extension supports virtualization**: guest states are stored in virtual machine control structures.
     - **OS requests directly trap VMM**: eliminates binary translation and guest OS modification.
     - VMM has root privilege, guest OS and user applications have non-root privilege.

	Full	Para	Hardware-assisted
Technique	Binary translation and direct execution.	Hypercalls.	OS requests trap to VMM without binary translation or paravirtualization.
Performance	Good.	Better in certain cases.	Fair.
Guest OS modification	No.	Yes.	No.
Is guest OS VMM independent?	Yes.	No.	Yes.
Guest OS privilege	Ring 1, non-root privilege.	Ring 0, root privilege.	Ring 0, non-root privilege.
VMM privilege	Ring 0, root privilege.	Below ring 0.	Below ring 0, root privilege.

- **Multi-tenancy**: enable dedicated software instances for each consumer, each with its own data and configuration.
  - Only possible because of virtualization.



Single-tenant - each cloud consumer has a separate IT resource instance.



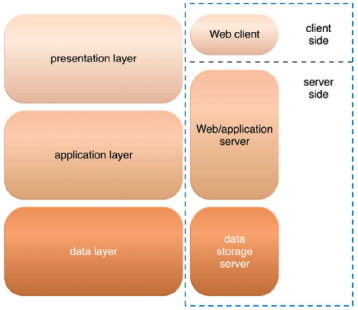
Multi-tenancy - a single instance of an IT resource on a cloud storage device serves multiple consumers.

(+) **Usage/tenant isolation**.
(+) **Data security**.
(+) **Scalable**.
(+) **Metered usage**: each tenant is only charged for features being used.
(+) **Data tier isolation**: databases & tables can be isolated or allowed as shared resources among tenants.

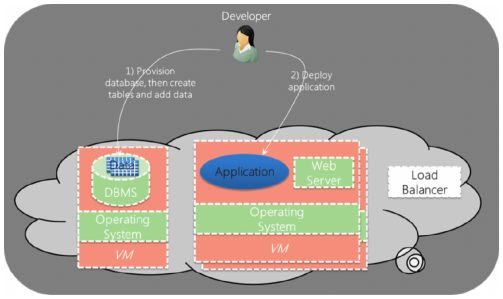
Applications & Programming

Applications & Paradigms

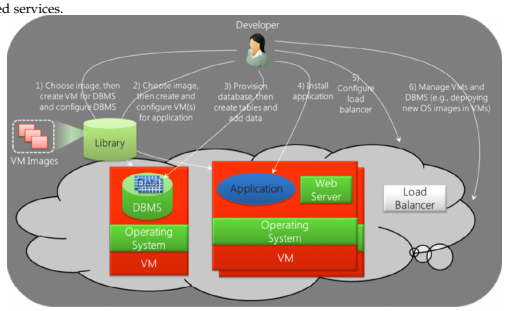
- **Cloud applications**:
  - Common features:
    - UI for accessing cloud resources.
    - Deployment management services for supporting elasticity and load balancing.
  - Cloud applications target *enterprise computing*.
    - Applications are suitable for cloud computing when they can be divided into as many independent tasks as possible and with minimum dependence among tasks.
    - Task dependencies → overhead and cost of communication between processing units.
  - Challenges:
    - **Performance isolation**: resource sharing → performance degrades when system is heavily loaded.
    - **Reliability**: server failures are common in data centers.
    - **Latency & bandwidth fluctuations**: due to sharing (virtualization & multitenancy).
    - **Data logging** required to monitor execution performance and resource usage → increased performance overhead.
- **Web application architecture**:
  - 3-tier architecture of web applications:
    - **Presentation layer**: contains the user interface.
      - Sits on both the client-side and server-side.
    - **Application layer**: contains implementation logic.
      - Web server receives client requests and retrieves the requested resources or generates web content based on application logic.
    - **Data layer**: consists of persistent data stores.



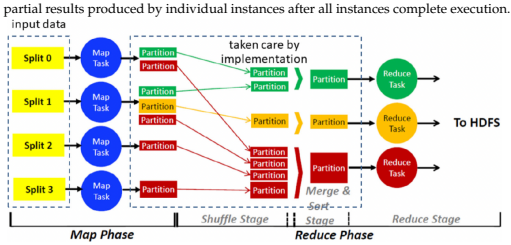
  - Architectural styles:
    - **Simple Object Access Protocol (SOAP)**: application protocol for web applications.
      - Defines a common web service messaging format for request and response message exchange.
    - **Representational State Transfer (REST)**: supports client communication with stateless servers.
      - Platform and language independent; supports data caching, and can be used in the presence of firewalls.
- **Cloud application development models**:
  - **Software-as-a-Service (SaaS)**:
    - Multi-tenanted applications.
    - Web access.
    - Centralized management of SaaS services.
    - Multi-device support.
    - Scalability under varying loads.
    - High availability.
    - API integration with other software.
  - **Platform-as-a-Service (PaaS)**:
    - All-in-one: same IDE to develop, test, deploy, host, and maintain applications.
    - Web access to development platforms.
    - Offline access for developers.
    - Built-in scalability.
    - Collaborative platform for developers.
    - Diverse client tools.



  - **Infrastructure-as-a-Service (IaaS)**:
    - Web access to resources.
    - Centralized physical resource management.
    - Elastic services and dynamic scaling.
    - Shared infrastructure across multiple users.
    - Preconfigured VMs.
    - Metered services.



  - **Function-as-a-Service (FaaS)**:
    - Enables scalable *event-driven applications*.
      - Developers focus on **application/business logic**.
      - Applications are divided into smaller, single-purpose (stateless) **lambda functions**.
      - Events trigger a lambda function, and die after execution.
    - Relieves cloud consumer of server management.
      - Consumer pays only when the code is executing.
    - **AWS Lambda** example usage:
      1. Create lambda service.
         (a) Business logic (code): lambda function(s).
         (b) Configuration: region to run, AWS credentials, etc.
         (c) Event (that triggers a lambda function).
      2. Deploy service created.
      3. Invoke/test service.
- **MapReduce**: supports distributed computing on large data sets on multiple machines.
  - Single Program, Multiple Data (SPMD) model.
  - Key idea:
    1. **Split** data into blocks, assign each block to an instance/process for parallel execution.
    2. **Merge** partial results produced by individual instances after all instances complete execution.

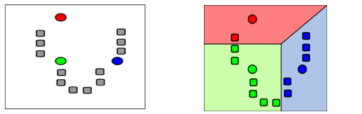


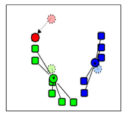
  - Stages:
    1. **Map**: a data file is partitioned into k smaller data files (splits).
    2. **Shuffle**: each split is evaluated by a map task (i.e., **Mappers**), producing data partitions (intermediate results) which may belong to different reduce tasks (i.e., **Reducers**).
    3. **Shuffle** (merge & sort): on arriving at a reduce task, partitions are merged into one file and sorted.
    4. **Reduce**: a reduce task evaluates the sorted file to produce the final result.
- **AWS components**:
  - **Amazon Elastic Compute Cloud (EC2)**: enables launching and managing server instances in Amazon's data centers.
  - **Amazon Simple Queue Service (SQS)**: handles message/work flows between other components in a system.
  - **AWS CloudFormation**: enables the creation and provision of AWS infrastructure deployments predictably and repeatedly.
  - **Amazon Simple Storage Service (S3)**: web storage service.
  - **Amazon Elastic Load Balancer**: automatically distributes incoming application traffic across multiple EC2 instances.
  - **Amazon CloudFront**: speeds up distribution of web content through a network of edge locations.

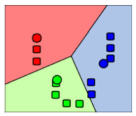
Examples

k-means clustering

- k-means: given n data points, divide them into k clusters.
  - Goal: find a partition which
    - Maximizes distance between inter-cluster data points.
    - Minimizes distance between intra-cluster data points.
    - Distance measure: typically **Euclidean distance** (a.k.a. 2-norm), $d(\vec{p}, \vec{q}) = \sqrt{\sum_{i=1}^{n}(q_i - p_1)^2}$.
  - Challenges:
    - Deciding on the value of k.
    - Initializing the k centroids.
  - Algorithm:
    1. **Map** phase:
       (a) Compute distances between an input point and all the previously obtained centroids of clusters.
       (b) Find the cluster with minimum distance.
       (c) Add the input point into the cluster with minimum distance.
    2. **Reduce** phase:
       (a) Receive all the points belonging to a particular cluster.
       (b) Compute new centroid.

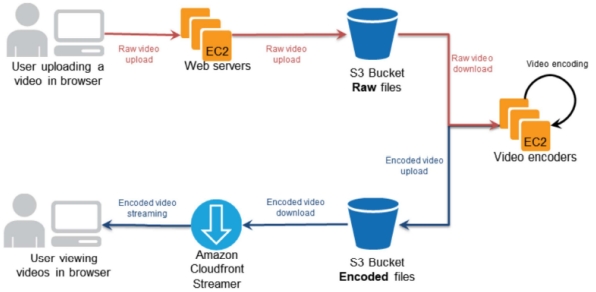




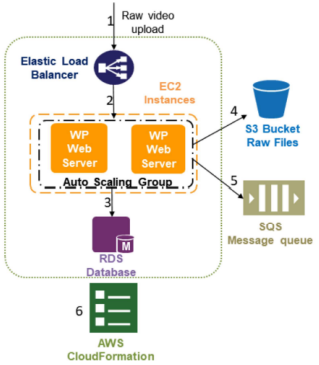


Video-sharing SaaS

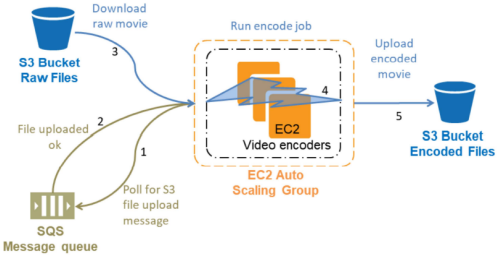
- 3 key components:



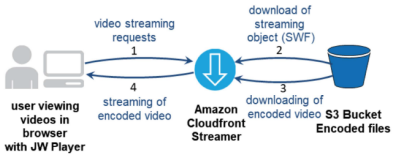
1. Users upload shared videos: raw video files are stored in Amazon S3 buckets.



2. Encode uploaded videos: compresses raw videos for more efficient storage on S3, and converts them into different streaming formats.



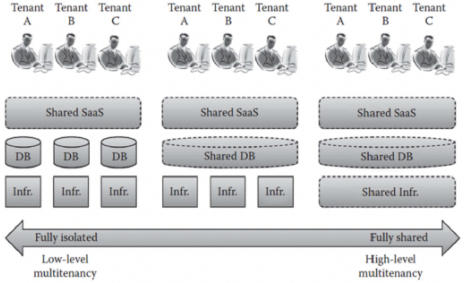
3. Stream shared videos to users: CloudFront distributes the encoded videos to different users.



- Performance bottlenecks:
  1. **Uploading**: >1 EC2 instances can be set up to handle multiple user uploads + ELB to load balance across available instances + S3 storage scales transparently with demand.
  2. **Encoding**: compute-intensive, auto-scaling can be used here.
  3. **Streaming**: CloudFront scales in a similar manner as a CDN.

Cloud software development

- SaaS is different from traditional software:
  1. **Pay-per-use**: in contrast to the traditional pay-for-full-license.
  2. **Zero infrastructure**: customers need not install the software.
  3. **Reduced business cost**: one SaaS application can be shared by multiple customers.
  4. **Automated updates**: updates can be performed by SaaS service providers.
  - Suitable for:
    - Consumers requiring on-demand software rather than full-term/licensing-based software.
    - Companies which do not have the financial means to invest in licensed software.
    - Applications with unpredictable and dynamic load.
  - Not suitable for:
    - Real-time processing: where fast data processing is needed.
    - Organizational data which is confidential and/or requires data localization.
    - Organizations with the capability to maintain on-premise applications.
- Perspectives of SaaS development:
  - Key challenges:
    1. Choosing **correct multitenancy level(s)**.
    2. **Governance and security** over user data.
  - Options:
    1. **Everything self-managed**: provider has full control → user data is more secure, but higher cost and lower resource utilization.
    2. **Shared infrastructure, self-managed platform**: higher infrastructure resource utilization, but overhead of platform maintenance; suitable for applications with lower requirements on data security.
    3. **Shared platform, self-managed infrastructure**: full control over user data, supports data localization and governance.
    4. **Everything shared/cloud-enabled**: higher application scalability (through dynamic scaling) & availability, but no governance over user data + risk of cross-tenant attacks across all layers.
- **Cloud-aware SaaS development**: using PaaS (over traditional software development) eliminates the reliance on licensed platforms + overhead in testing, development, scaling, and maintenance of the application. Other considerations include:
  - **Delivery model**: has to satisfy customer requirements, e.g., security level & ROI.
  - **Deployment model**: public v. private, tradeoff between maintenance overhead & security.
  - **Degree of sharing/multitenancy**: higher multitenancy → higher resource utilization, poorer data security.

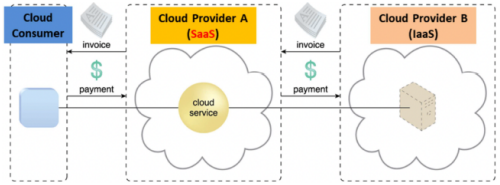


  - **Dynamic scaling**: especially crucial for SaaS applications, satisfied by PaaS + load balancing.
  - **Availability**: satisfied by the distributed setup managed by the PaaS provider.
  - **Database-level multitenancy**: multiple options, between sharing db instance v. sharing table v. sharing schema.
    - NoSQL is generally preferred for SaaS applications to achieve scalability + availability at the db level.
  - **Need to monitor & prevent** tenant misbehaviour, failure, security attacks, and SLA violations.

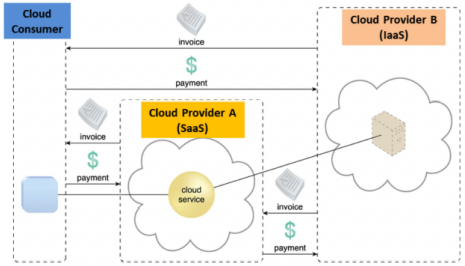
Cloud Management

Pricing models

- **Pricing models**: define the unit cost of resource usage according to usage cost metrics.
  - Influencing factors:
    1. **Overhead in design, development, deployment, and operation** of cloud services.
    2. **Opportunities to reduce expenses** via IT resource sharing and data center optimization.
    3. **Other business costs**: market competition & regulatory requirements.
  - Variables:
    1. Cost metrics & associated prices: depends on either **on-demand** v. **reserved allocation**.
    2. Fixed and variable rates.
    3. Discounts for higher volumes.
    4. Payment options and schedules: prepaid v. postpaid v. installments.
  - For SaaS and PaaS:
    - **Integrated pricing**: consumer pays the SaaS provider, who in turn pays the IaaS provider for hosting.



    - **Separate pricing**: consumer pays SaaS provider for the service + IaaS provider for hosting the service.



- **AWS EC2 instance types**: VMs are diversified to support different types of applications and consumer requirements.
  1. **General purpose**: t2, m4.
  2. **Compute optimized**: c3, c4.
  3. **Memory optimized**: x1, r3.
  4. **Storage optimized**: i3, d2.
  5. **Accelerated computing**: p2.

  Each instance type is further divided into subtypes, based on the **number of virtual CPUs** and **memory availability**.
  - e.g., nano, micro, small, 10xlarge, 16xlarge, etc.
- **AWS pricing model**: differs across regions, depends on operating costs.
  1. **On-demand**: multitenant, pay based on usage (seconds).
  2. **Reserved**: onetime upfront cost + pay based on usage (lower rate than on-demand).
  3. **Spot**: consumers bid for unused capacity (lower rate than on-demand), instance automatically terminated if bidding price is lower than market rate.
  4. **Dedicated**: instance runs in a VPC/hardware dedicated to a consumer, physically isolated from other instances.

On-demand Instances	Reserved Instances	Spot Instances	Dedicated Instances
• Pay as you go • Starts from 0.03/Hour	• Onetime upfront + Pay as you go • $56 for 1 year term and then $0.01/Hour	• Requested Bid Price and Pay as you go • $0.005 /Hour as of today at 9 AM	• Standard and Reserved • Multi-Tenant Single Customer • $10/Region + 0.105/Hour
For Spiky Workloads	For Steady State Workloads	For Time-insensitive workloads	For Regulatory and Compliant Workloads

Modelling TCO

- **Business cost metrics**:
  1. **Upfront costs**: capital expenses, initial investment, deployment and administration of IT resources.
  2. **Ongoing costs**: operational expenses for running and maintaining IT resources.
  3. **Additional costs**:
     (a) **Cost of capital**: cost incurred to raise funds for IT resources.
     (b) **Sunk costs**: prior investment on existing IT resources.
     (c) **Integration costs**: effort needed to inter-operate IT resources in new environment.
     (d) **Locked-in costs**: cost of moving from one cloud provider to another.

  Typically, business costs are higher for on-premise than for cloud-based infrastructure.
- **Cloud usage cost metrics**:
  1. **Network usage**: inbound/outbound + intra-cloud data transfer, typically free or charged at different rates.
  2. **Server usage**: pay-per-use or upfront cost, depending on the pricing model.
  3. **Cloud storage device**: storage space + I/O data transferred, the latter may not be charged.
  4. **Cloud service**: cost varies based on subscription duration + number of users and txs.
- **Total cost of ownership (TCO)** = datacenter capex + datacenter opex + server capex + server opex.
  - **Capital expenses (capex)**: upfront investment, depreciated over a certain time frame.
  - **Operational expenses (opex)**: recurring cost of running the datacenter.