Karlsruhe Institute of Technology

Software Engineering II

Prof. Dr. Ralf H. Reussner

Topic 08

Cloud Computing & Cloud Architecture

DEPENDABILITY OF SOFTWARE-INTENSIVE SYSTEMS
KASTEL – INSTITUTE OF INFORMATION SECURITY AND DEPENDABILITY

dsis.kastel.kit.edu

Overview on Today's Lecture

Karlsruhe Institute of Technology

Content

- Characteristics of Cloud Computing
- Delivery Models
- Deployment Models
- Virtualisation
- Multi-tenancy
- Cloud Architecture: Example and Principles

Learning goals: participants –

- able to explain the differences between cloud computing and hosted resources regarding used technologies, software delivery model
- able to explain the economics of cloud computing
- able to explain the map-reduce programming model
- able to explain and apply architectural principles of cloud software

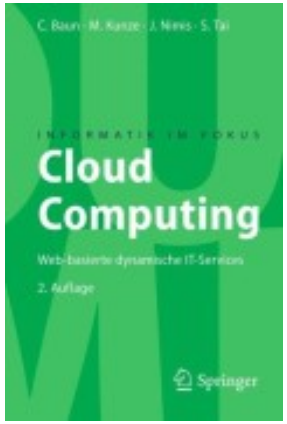
Slides

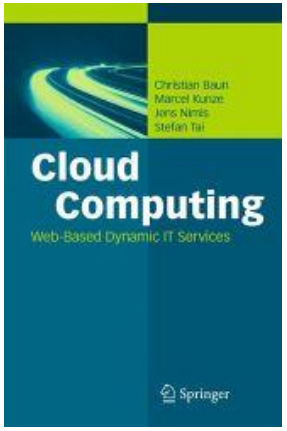
- Slides of this course are partially provided courtesy of
  - Prof. Dr. Stefan Tai

Cloud Computing: Definition

*"Building on compute and storage virtualization, cloud computing provides scalable, network-centric, abstracted IT infrastructure, platforms, and applications as on-demand **services** that are billed by consumption."*

– C.Baun, M.Kunze, J.Nimis, S.Tai: Cloud Computing, Informatik im Fokus, Springer 2009-2011





Read Online:
http://www.springerlink.com/content/978-3-642-18436-9

What are essential characteristics of cloud services?

There are
five essential characteristics
of cloud services

–[NIST]

[NIST]: http://csrc.nist.gov/groups/SNS/cloud-computing/

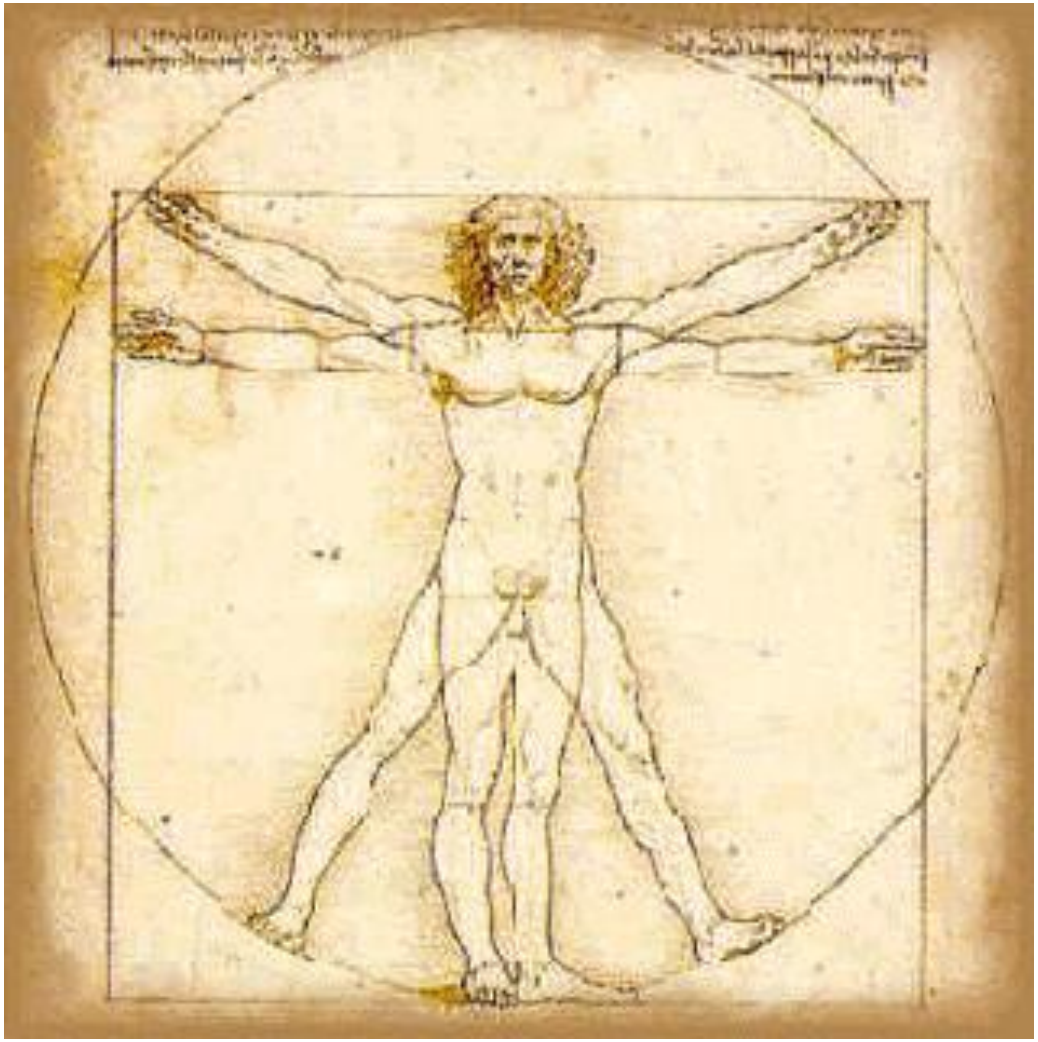
Elastic Scalability

On-demand Self-Service

Ubiquitous Network Access

Resource Pooling

Measured Service



Key Points

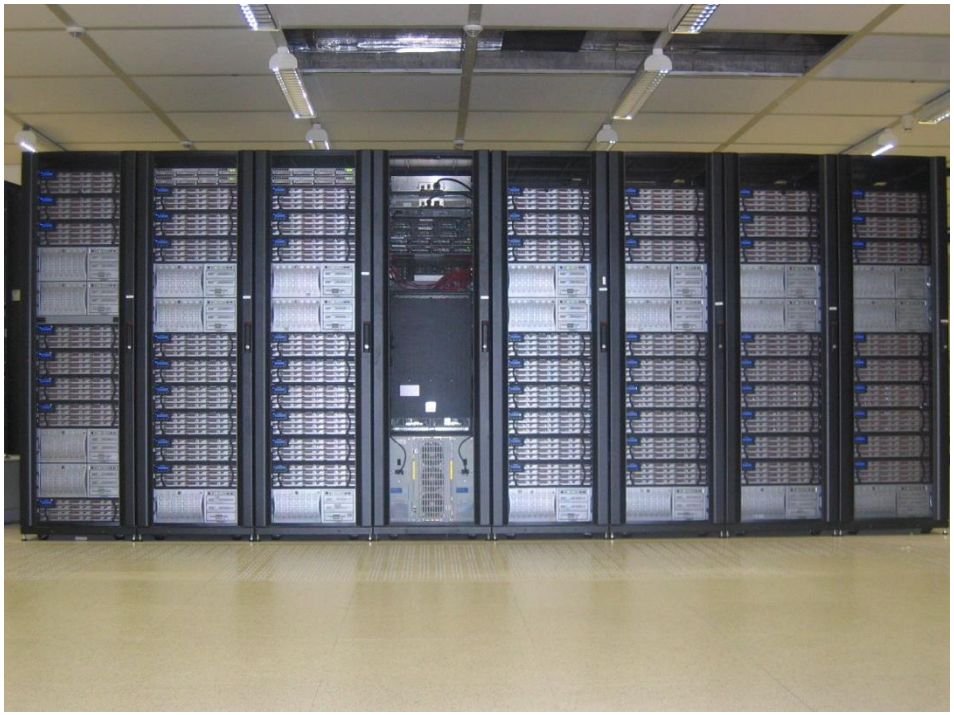
- The illusion of infinite IT-resources, which are
  - always available,
  - network-accessible
  - as on-demand services that
  - do not require ownership, prior reservation, etc
- Pay for use business model

Further reading:
“Above the Clouds: A Berkeley View of Cloud Computing”:
http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf

What are Resources?

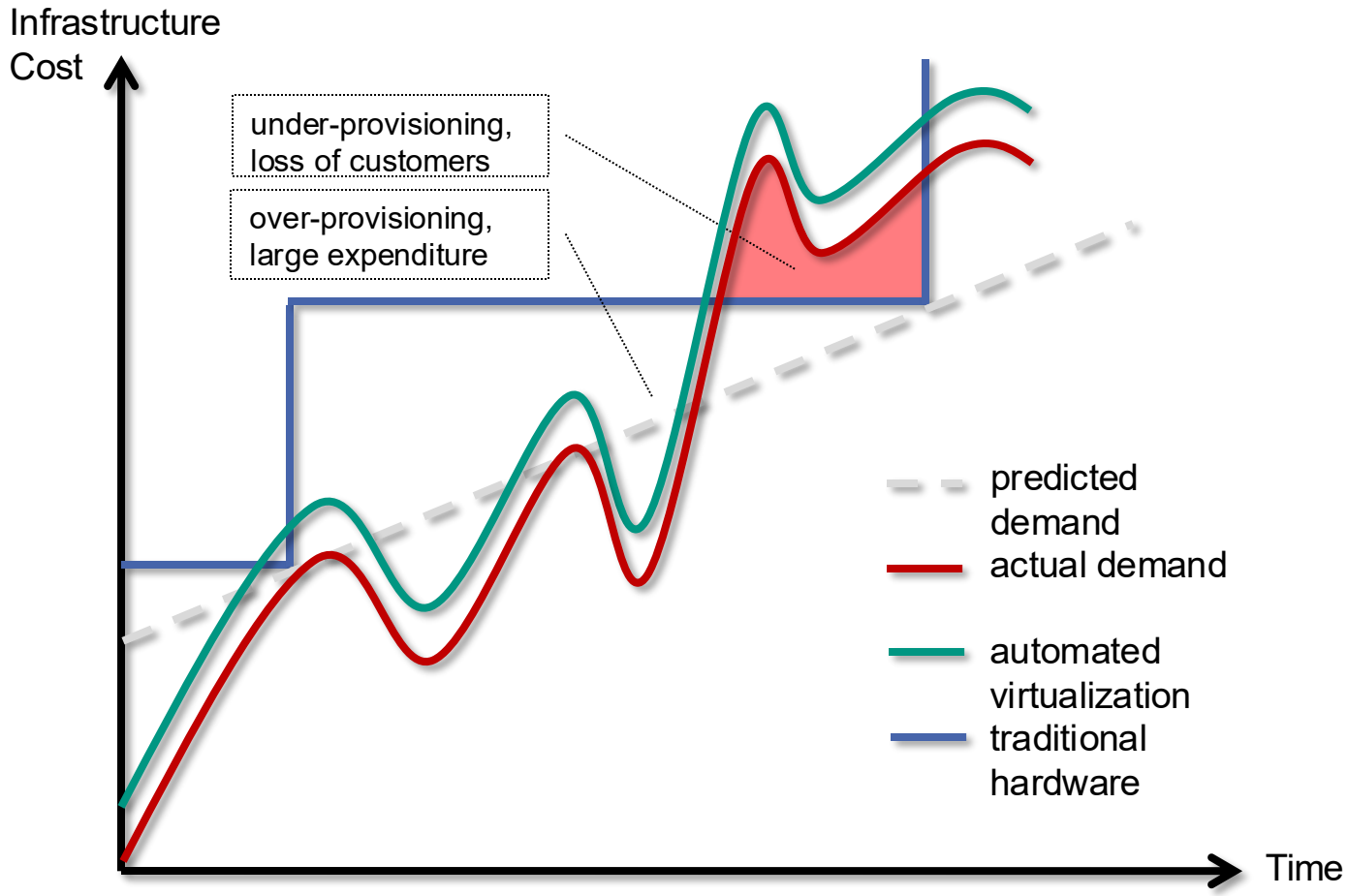
- In computing, in general
  - Compute Power
  - Storage
  - Network
- But also
  - **$**
  - Energy
  - Real-Estate
  - **$ $ $**
  - Humans (Manpower, Human/Crowd Intelligence)
- And
  - Software Apps
  - Data
- …

Problem: Efficient Use of Resources



- Concurrent use by various customers (tenants) and scheduling systems
- Varying requirements wrt. operating system, software, hardware, etc.
- Unpredictable load, especially often in Web systems

Problem of Resource Provisioning



- How accurately can we do the resource planning to follow demand?
- Fixed cost (expensive!) versus variable cost
- Cloud computing solves the resource provisioning problem

Scalability and Elasticity

Scalability is the ability of an application to counter *high load* through the utilisation of *more ressources*.

Elasticity is the ability of a *ressource management system* to provide *ressources* according to *varying demand*.

This means:

- providing more ressources, as the load increases. (to deal with high load scenarios)
- de-allocating ressources, as the load decreases. Important to avoid wasted costs, pay per use)

[Herbst, Kounev, Reussner:
Elasticity in cloud computing: What it is, and what it is not, ICAC 2013]

Multi-Tenancy and Partitioning

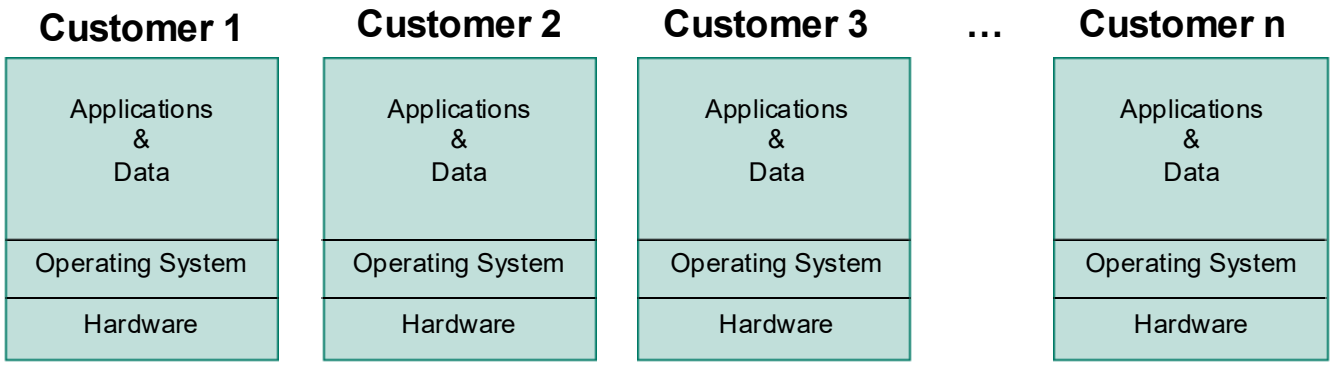
Multi-Tenancy

- Isolation of workloads
- Separation of customers
- Customers gain administrative privileges

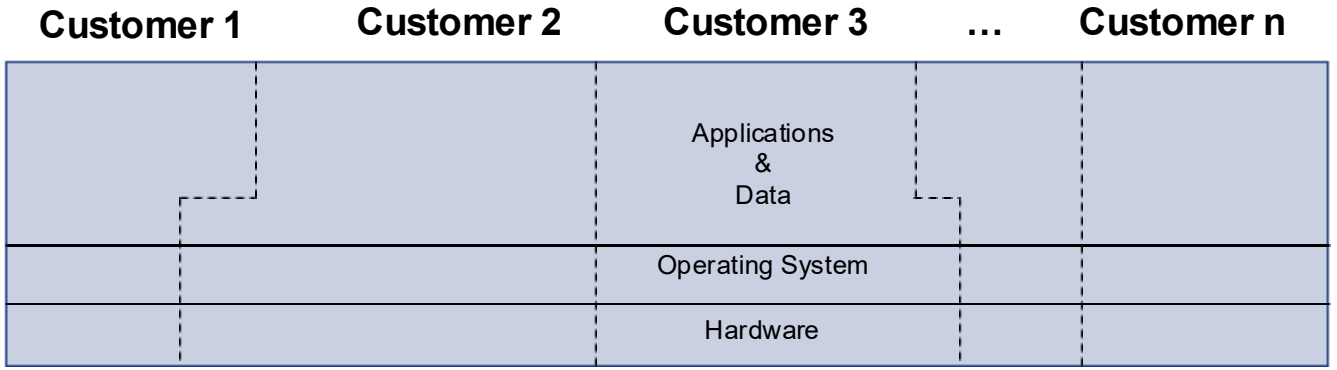
Partitioning of Resources

- Classical solution: Separation of servers
  => *Physical Resource Sets* (PRS)
- Cloud solution: Separation of services
  => *Virtual Resource Sets* (VRS)

Single-Tenant vs. Multi-Tenant Architecture



single tenant architecture



multi tenant architecture

Multi-Tenancy

- Multi-tenancy requires an architecture that *maximizes* the sharing of resources across multiple tenants, but that is still able to *differentiate* data belonging to different customers
- Instead of *customizing* applications in the traditional sense, each customer [in a multi-tenant architecture] uses metadata to *configure* the way the application appears and behaves for its users
- The challenge for the SaaS architect is to ensure that the task of *configuring* applications is simple and easy for the customers, without incurring extra *development or operation costs* for each configuration.

Virtualization

"Virtualization is the process of presenting computing resources in ways that users and applications can easily get value out of them, rather than presenting them in a way dictated by their implementation, geographic location, or physical packaging.

In other words, it provides a logical rather than physical view of data, computing power, storage capacity, and other resources."

– J. Eunice, Illuminata

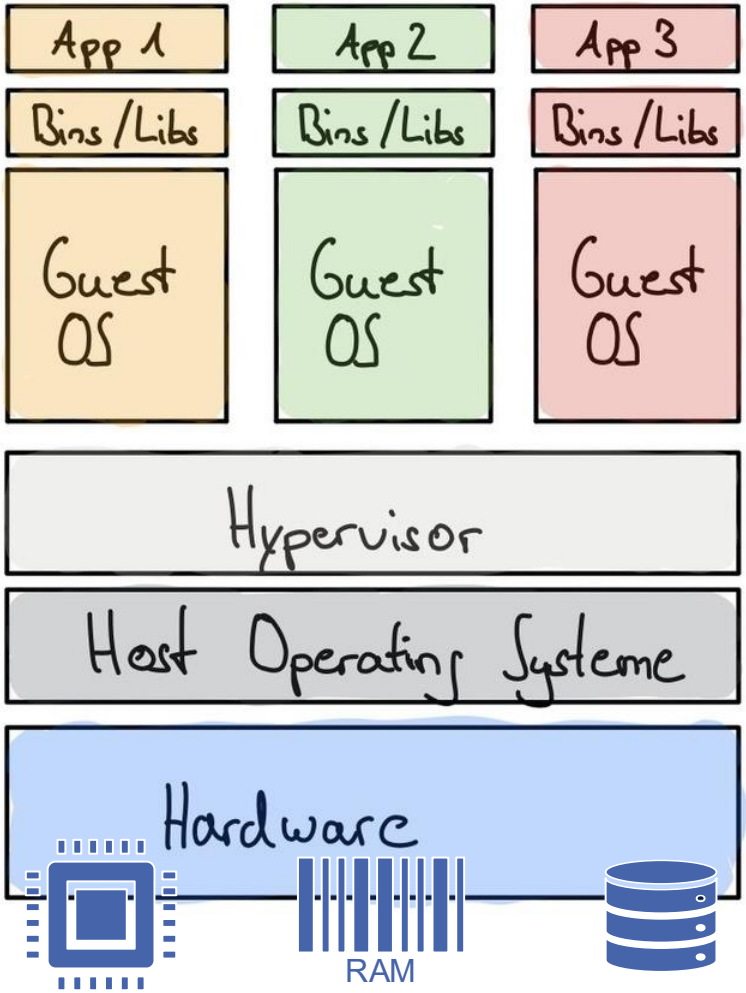
Virtualization Ecosystem

PlayOnMac

Microsoft
Hyper-V

and many more …

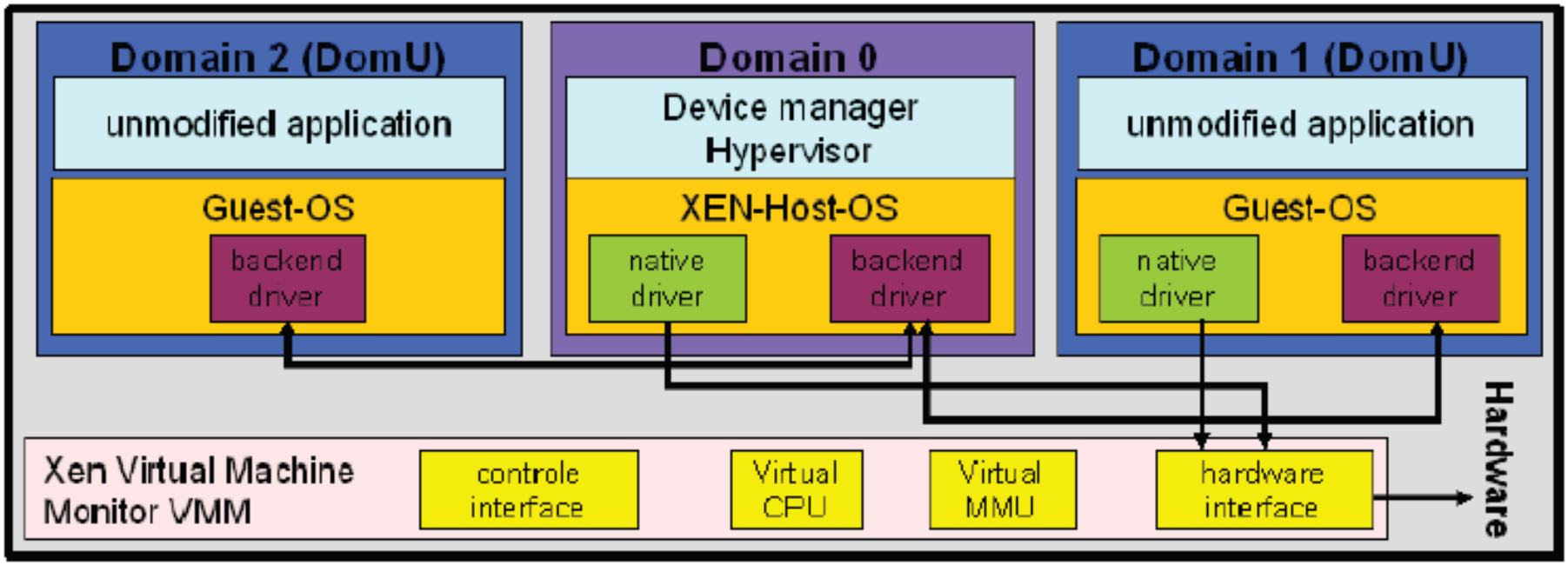
Virtual Machines using Hypervisor



- **Abstract view rather than physical view on infrastructure**
- **Hypervisor = control program**
- **Partitioning:** Multiple OS on single server
- **Isolation:** No side effects between VMs

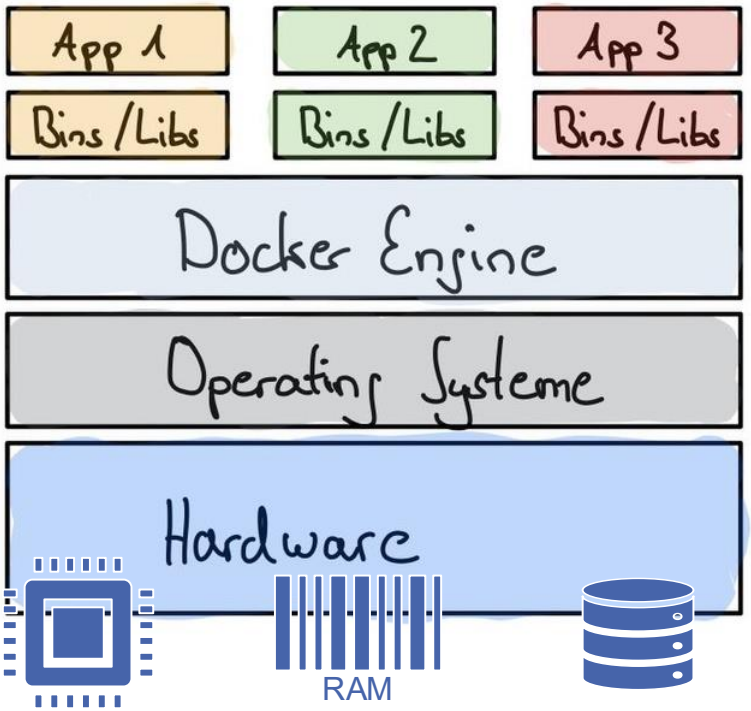
"Docker vs VM visualised" by Drinkler (CC-BY-SA-4.0)
https://commons.wikimedia.org/wiki/File:Dockervsvm.jpg

Paravirtualization



No emulated HW layer, but guest OS use hyper calls (hypervisor interface calls)

Containerization



- Container is a file system
  - Application code
  - Runtime environment
  - System tools
  - System libraries
- Container directly run on host operating system
- Container engine supports creating, linking, and managing containers

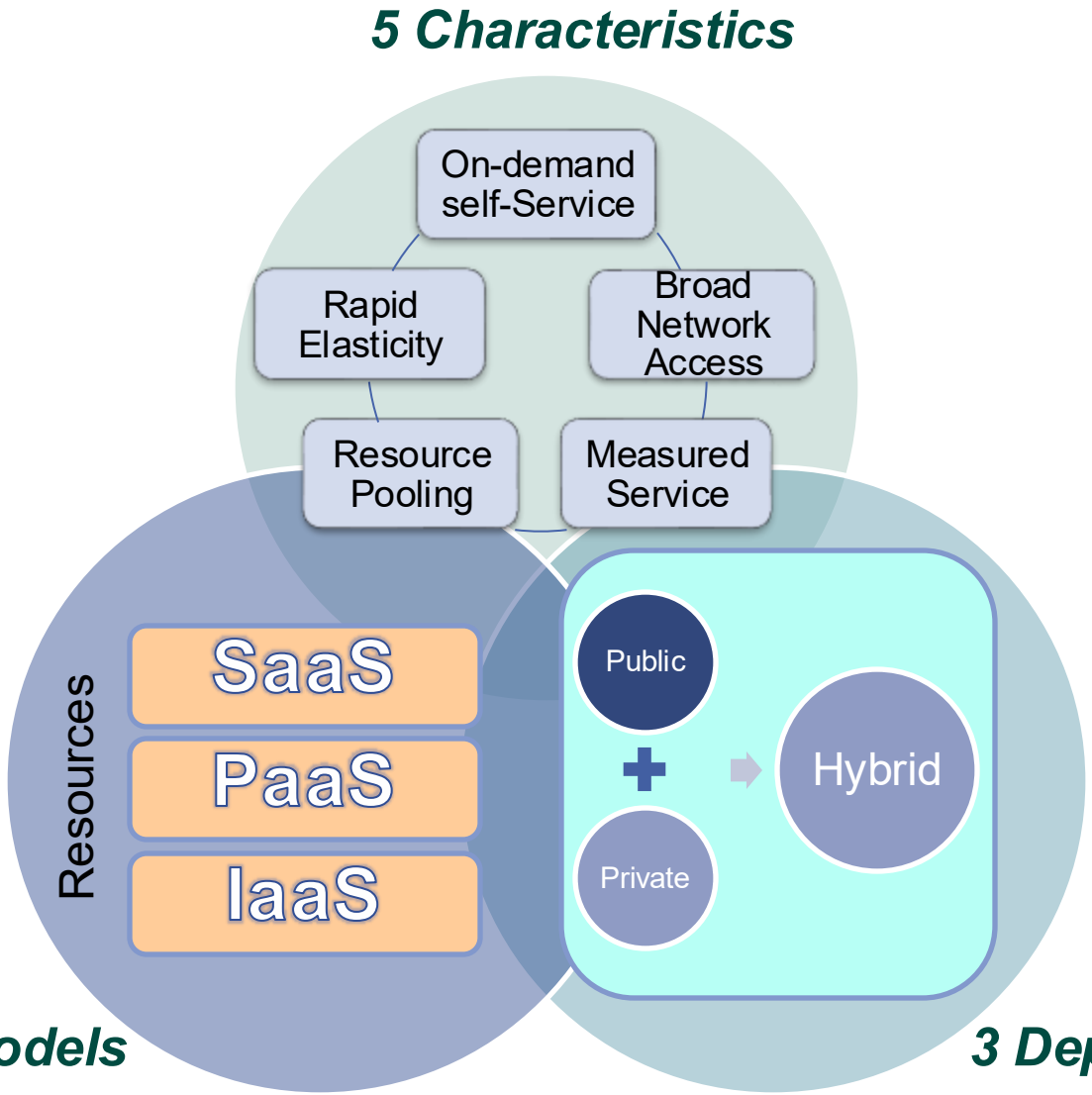
More in next lecture

"Docker vs VM visualised" by Drinkler (CC-BY-SA-4.0)
https://commons.wikimedia.org/wiki/File:Dockervsvm.jpg

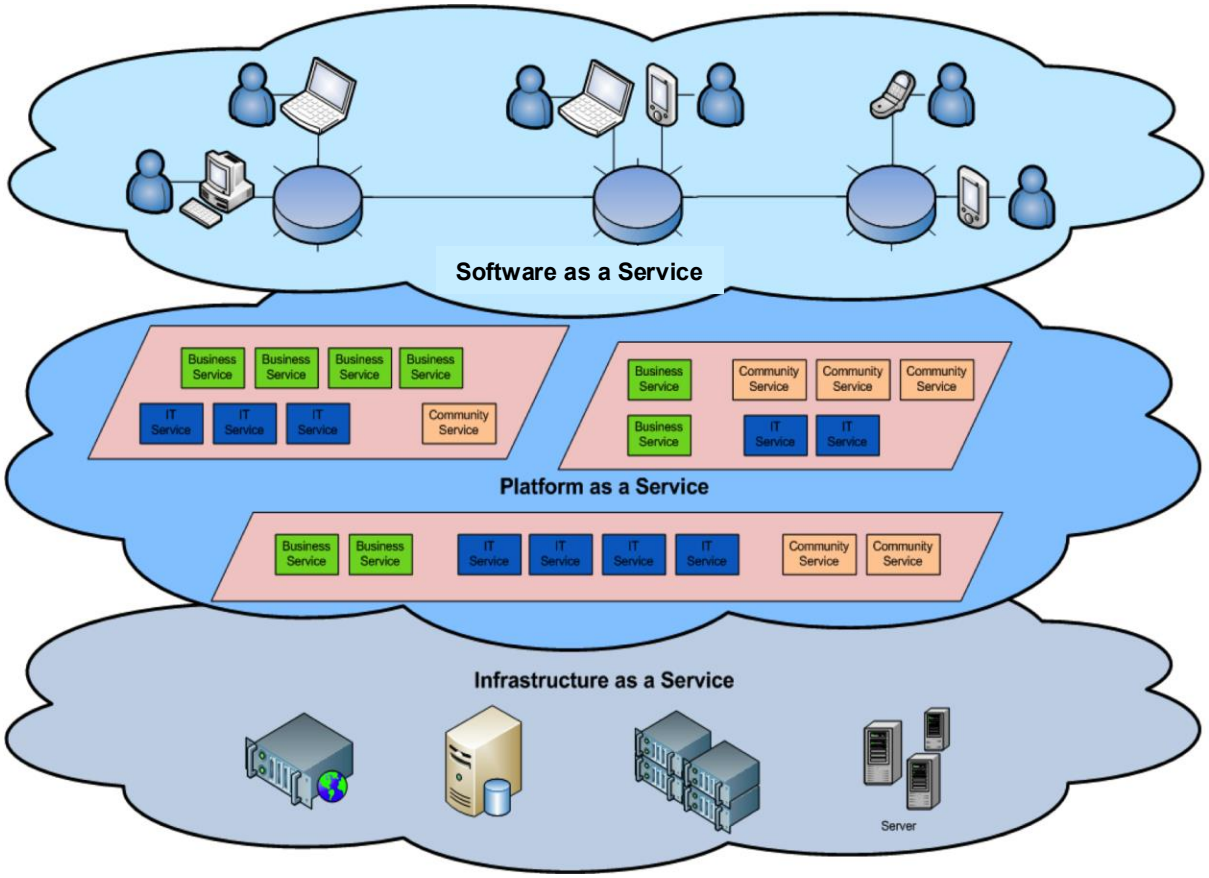
Advantages of Virtualization

- **Consolidation: Improved energy efficiency**
  - Actual blade servers are able to host up to 100 VM images
- **Availability**
  - Migration of VMs between blades on-line, without service interrupt
- **Quality of Service (QoS)**
  - Service Level Agreements (SLA)
- **Overbooking of resources (Thin provisioning)**
  - Decoupling of demand and resource provisioning
  - De-duplication of storage: Store identical pages only once
- **Logical view on resource pools improves management**
  - Capacity management rather than infrastructure management
  - Automation of infrastructure operation and service delivery
- **Almost no performance penalty with current technology**
  - Multicore
  - Hardware virtualization (Intel VT-x (prev. Vanderpool), AMD-V)

Cloud Computing Conceptional View [NIST]



Service Delivery Models



SaaS

Web Applications like Google Apps, Salesforce etc.

PaaS

Development or execution environments like e.g. Google App Engine, Windows Azure

IaaS

Offerings like Google Storage, Amazon Web Services etc.

Infrastructure as a Service (IaaS) [NIST]

- The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources…
- where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications.
- The consumer does not manage or control the underlying cloud infrastructure, but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components *(e.g., host firewalls)*.

Examples

- Amazon Web Services: Amazon Elastic Compute Cloud (EC2), Amazon Simple Storage Service (S3)
- Microsoft Azure Virtual Machines
- Google Cloud Platform: Google Compute Engine, Google Storage
- IBM SmartCloud Foundations

More comprehensive list at [https://en.wikipedia.org/wiki/Cloud-computing_comparison]

Platform as a Service (PaaS) [NIST]

- The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications…
- created using programming languages, libraries, services, and tools supported by the provider.
- The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment.

Examples

- Microsoft Azure Web Apps
- Google App Engine (also part of Google Cloud Platform)
- IBM SmartCloud Services

Software as a Service (SaaS) [NIST]

- The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure.
- The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface.
- The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.

Examples

- Salesforce.com
- IBM SmartCloud Solutions

Key Characteristics of SaaS

- Network-based access to, and management of, commercially available software
- Activities that are managed from central locations rather than at each customer's site, enabling customers to access applications remotely via the Web
- Application delivery that typically is closer to a one-to-many model (single instance, multi-tenant architecture) than to a one-to-one model, including architecture, pricing, partnering, and management characteristics
- Centralized feature updating, which obviates the need for downloadable patches and upgrades

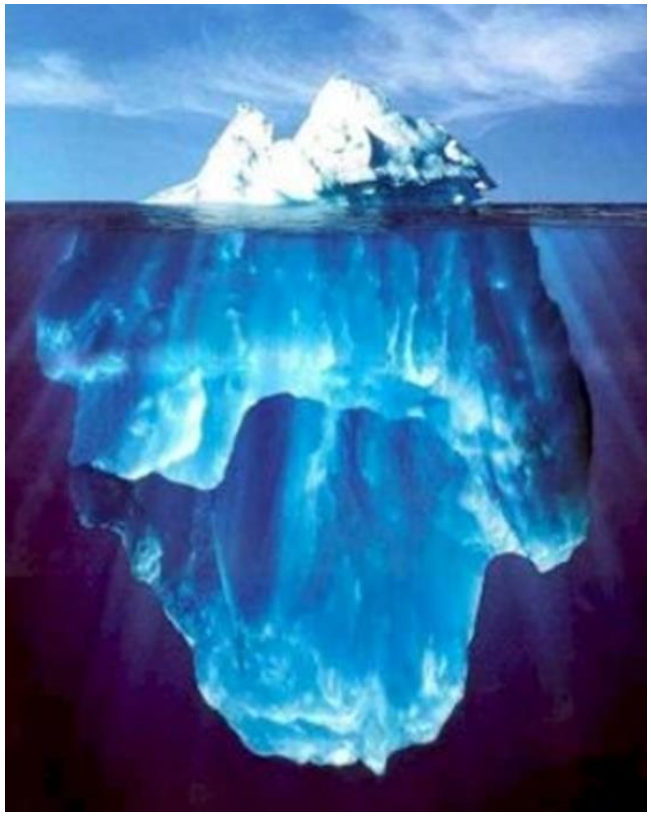
Boundaries are often blurred

- Amazon S3 is more than a harddisk would offer but seen as IaaS (but could also be seen as PaaS, through its high-level database access services)
- Salesforce offers services to end-users, and is hence seen as SaaS (but could also be seen as PaaS, as services are highly configurable (even programmable through script language))

Guidelines:

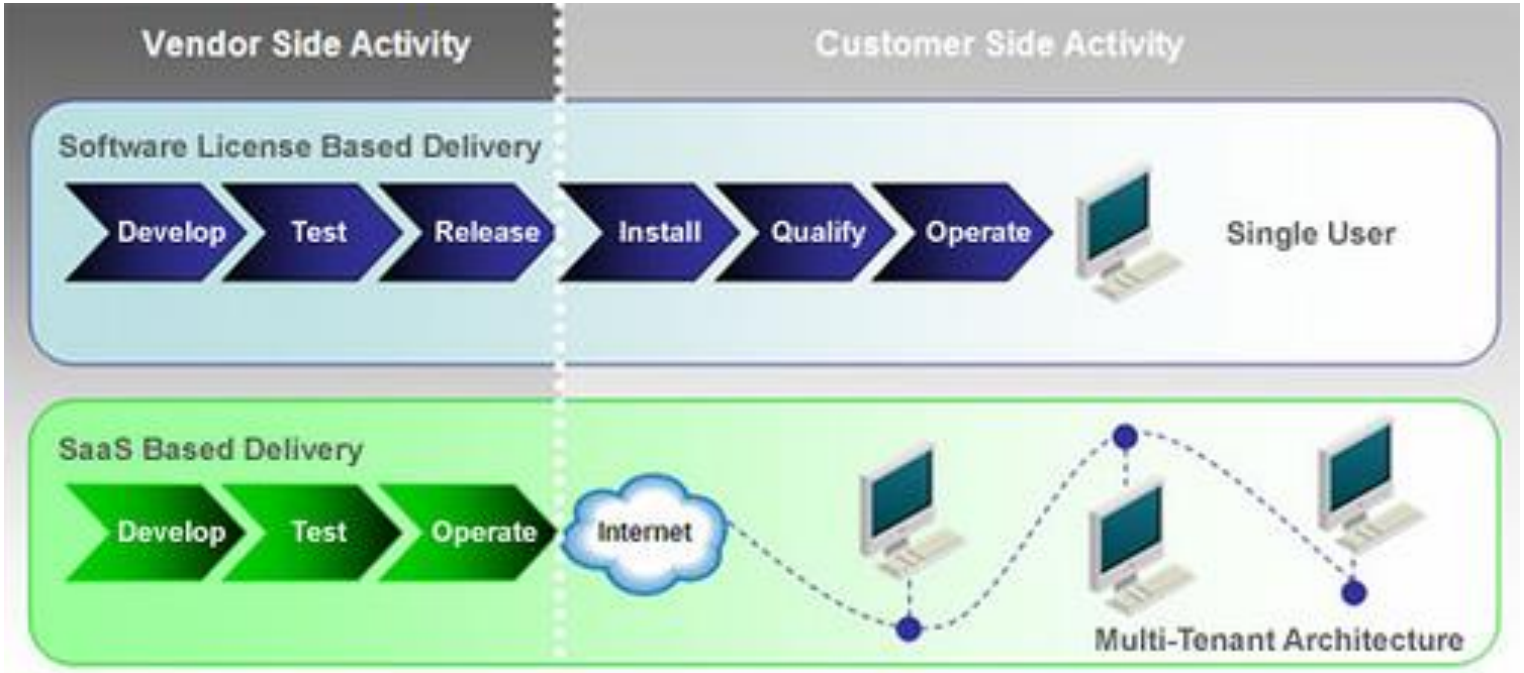
- If service is centered around classical ressources (compute, storage) → IaaS
- If service adresses developer (and not classical end-users) → PaaS
- If service offered end-user usable functionality → SaaS

Total Cost of Ownership (TCO) of License-based Software Delivery

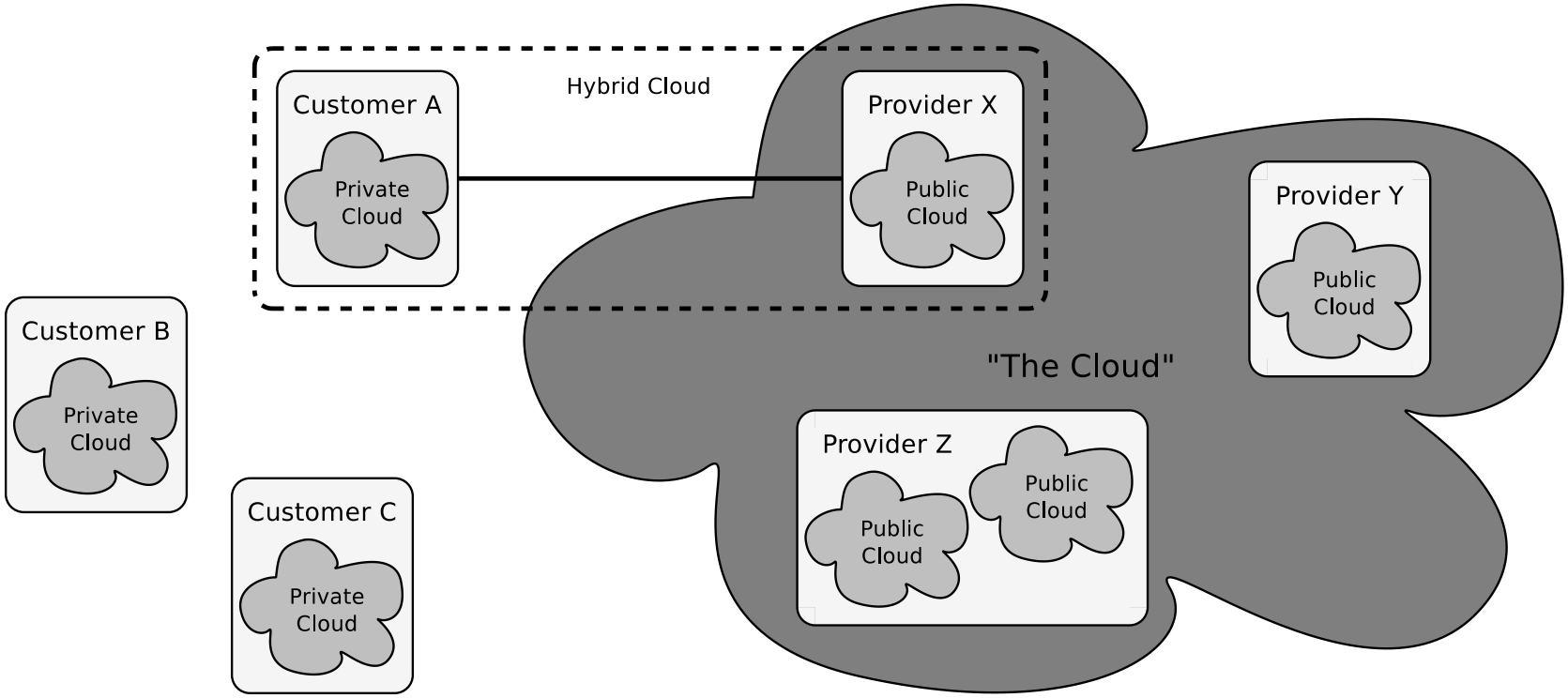


- Software licenses
- Hardware costs
- Software updates, update contracts
- Patches and fixes
- Backups
- Incident & Change Management
- Systems monitoring and management
- Ensuring availability, reliability, etc
- Human resources
- Energy costs
- etc.

Software License-based Delivery vs. SaaS

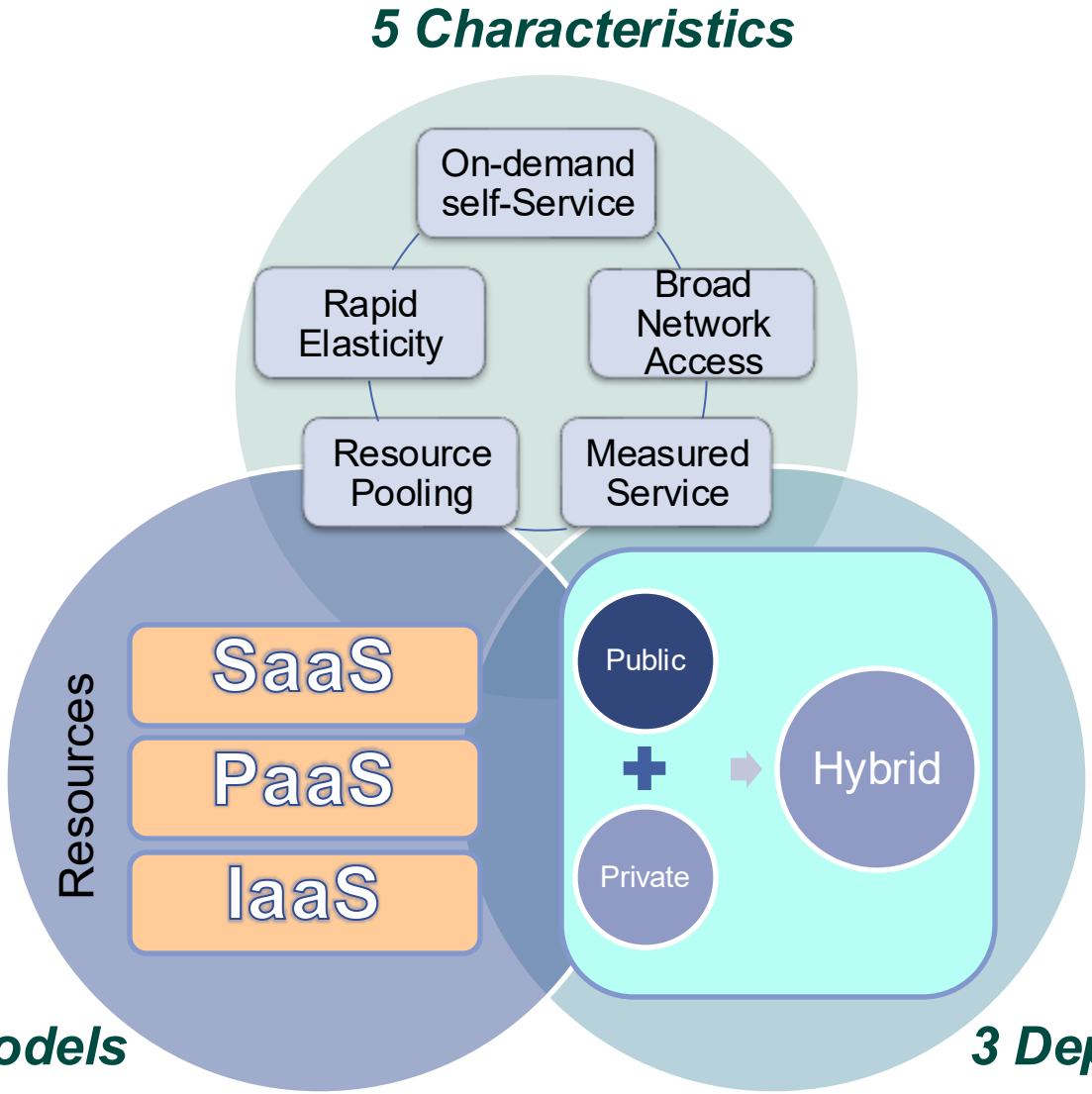


Service Deployment Models



- Private cloud: Customer and provider belong to the same organization
- Public cloud: Customer and provider belong to different organizations
- Hybrid cloud: Combination of private and public cloud
- Community cloud: Stakeholders share resources to achieve common goal

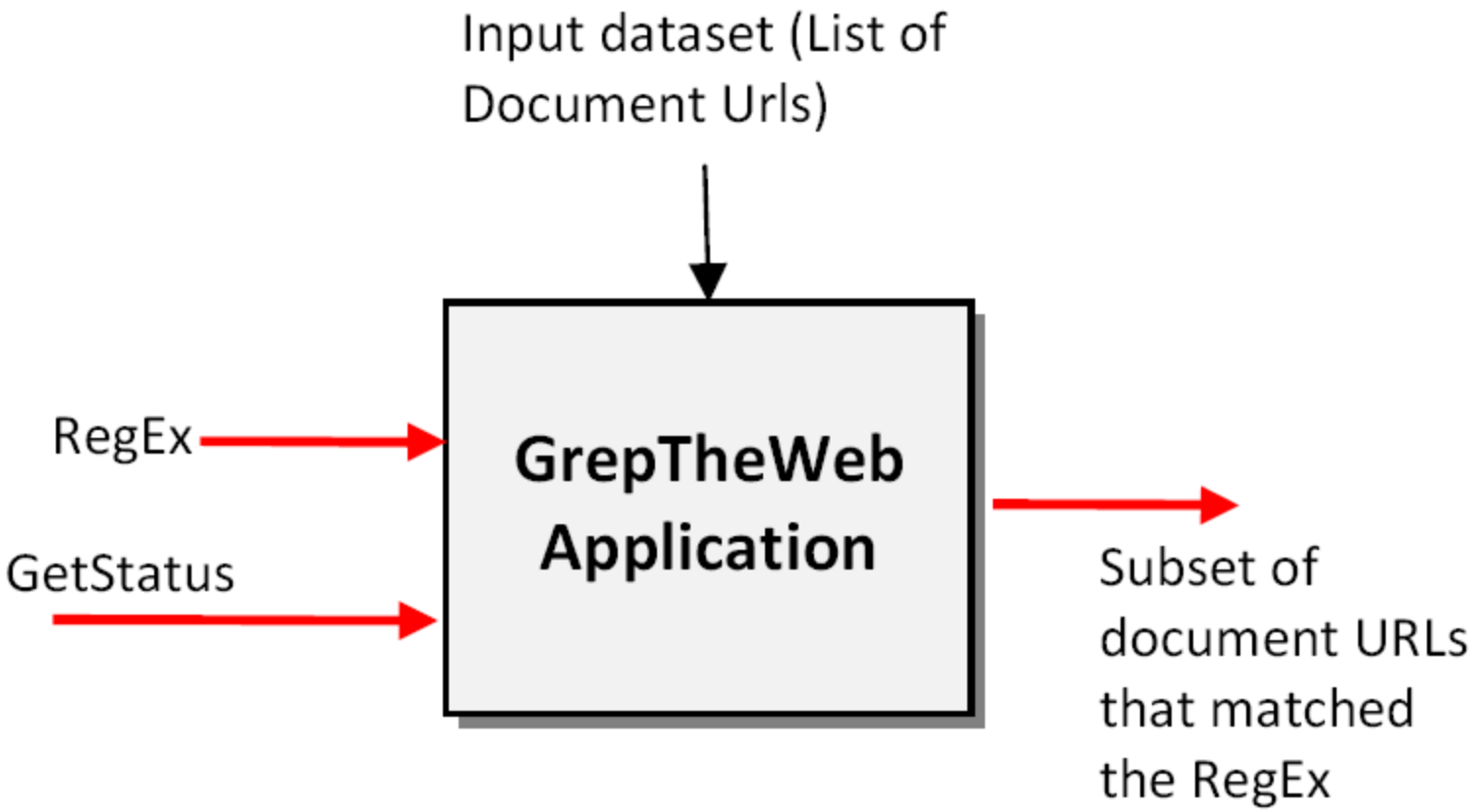
Cloud Computing Conceptional View [NIST]



"Grep The Web"

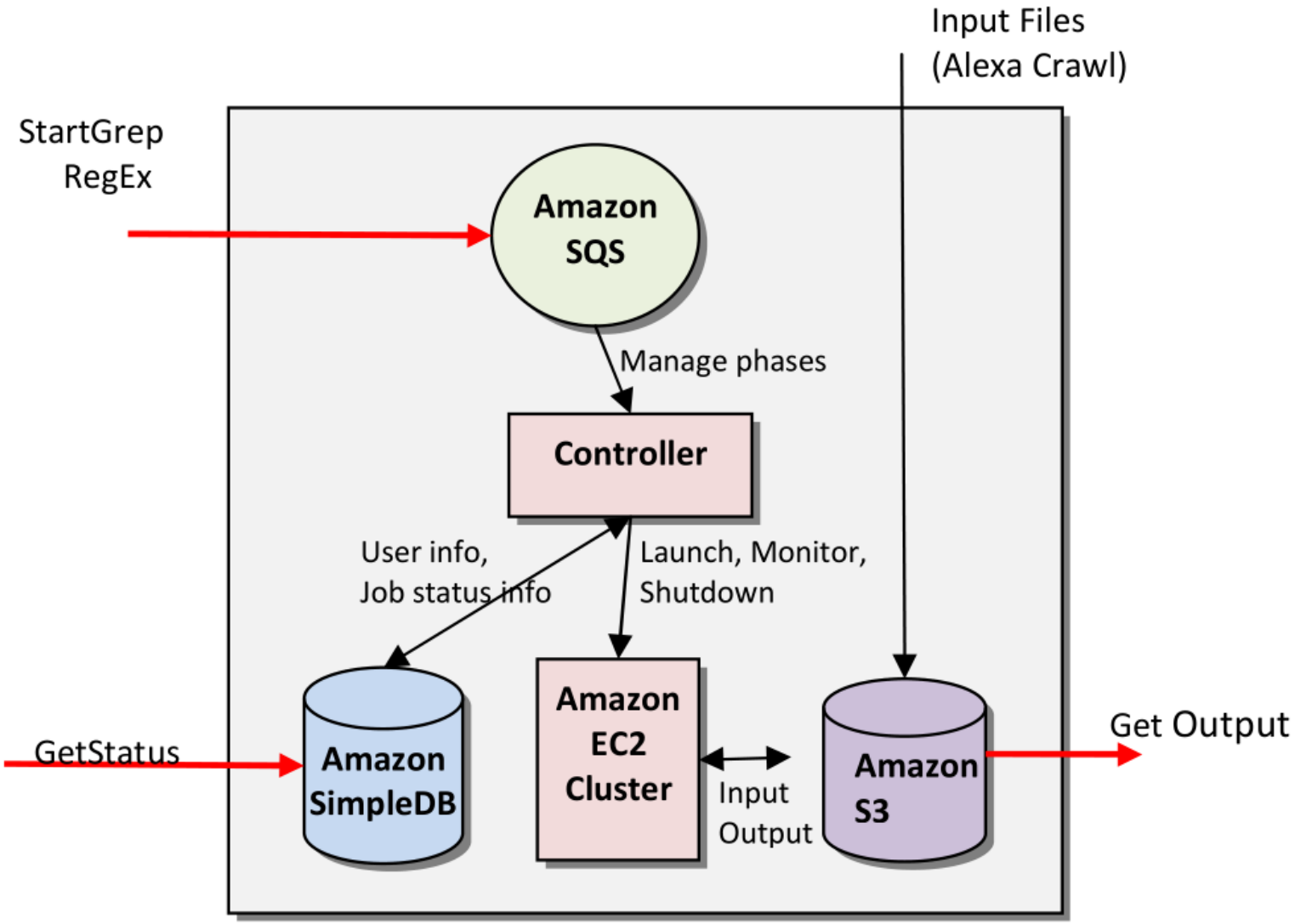
[AmazonGrepTheWeb] J. Varia „Cloud Architectures“, White Paper, Amazon, 2008. Available online at http://technicalscope.co/upload/aws-cloud-architectures_1427123268.pdf

GrepThe Web – Zoom Level 1



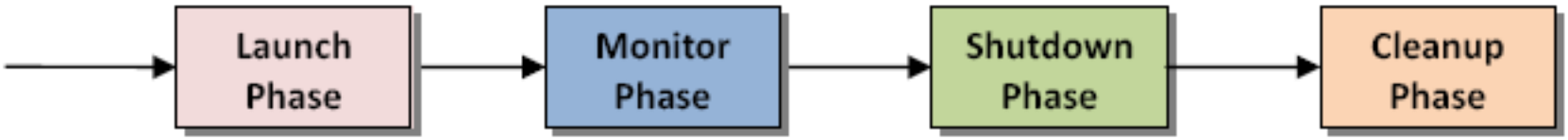
Source: amazon.com

GrepThe Web – Zoom Level 2



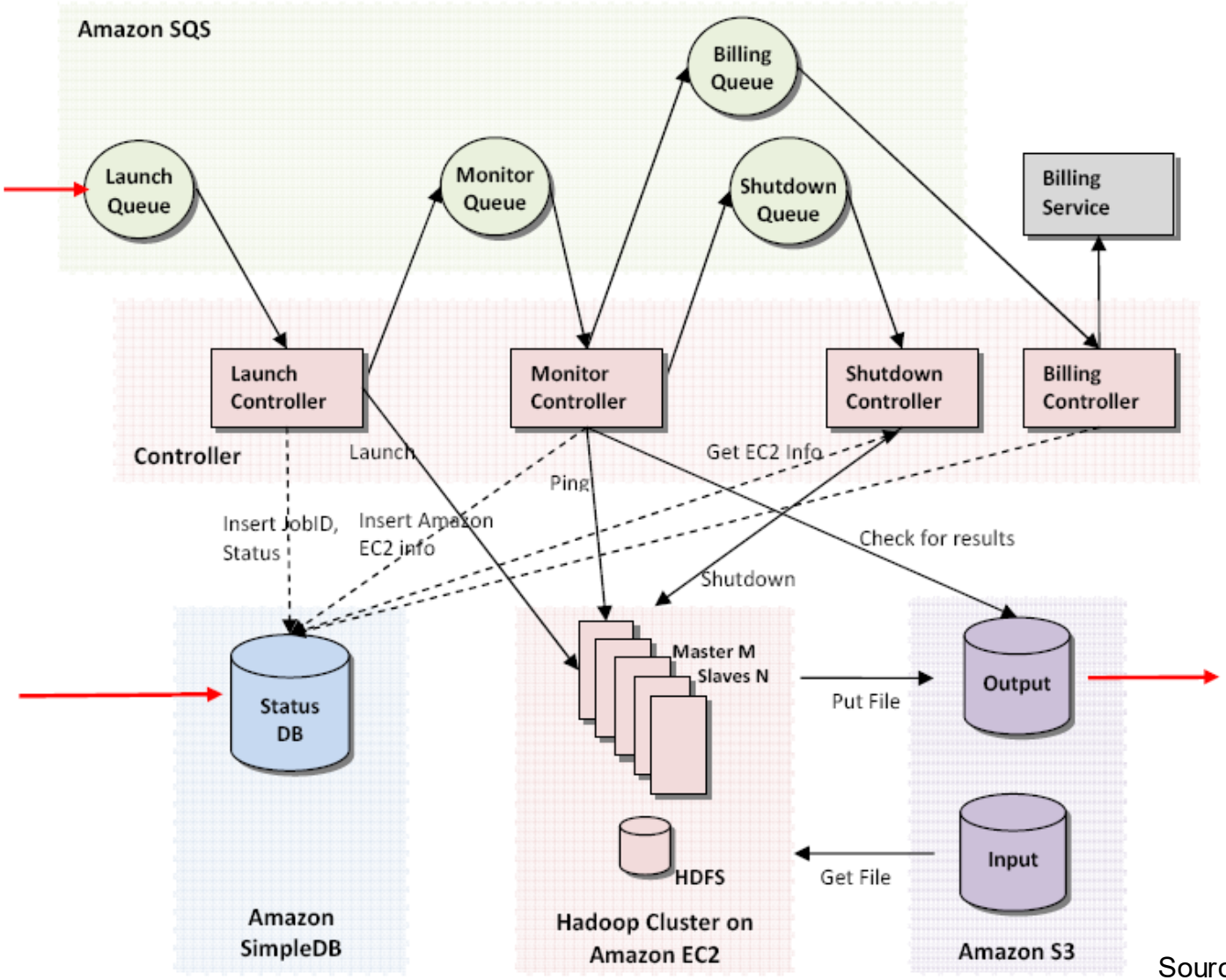
Source: amazon.com

Phases of GrepThe Web Architecture



Source: amazon.com

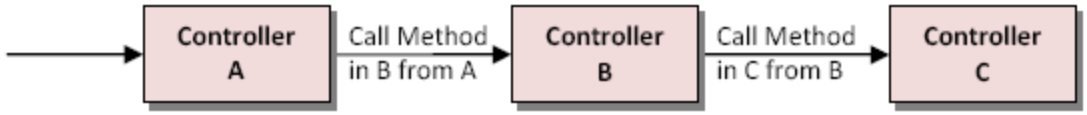
GrepThe Web – Zoom Level 3



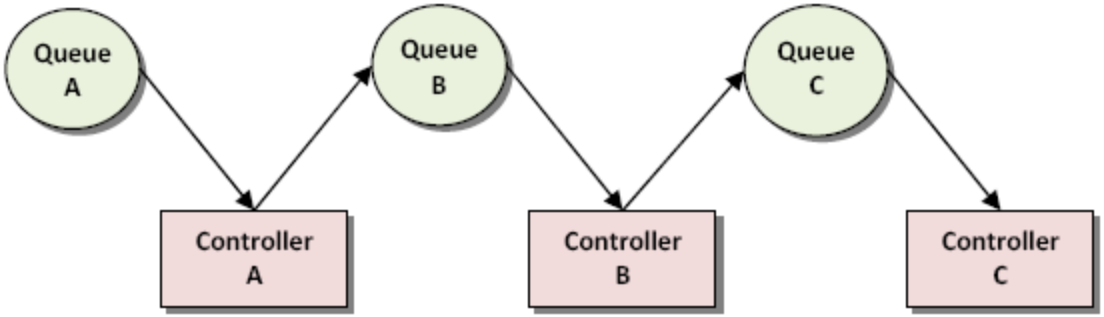
Source: amazon.com

Using Queues for Loose coupling

- Principle: For better manageability and high-availability, make sure that your components are **loosely coupled**
- Here:
  - Loose coupling by message queues.
  - Components can fail independently, queues buffer until back up



Tight coupling (procedural programming)

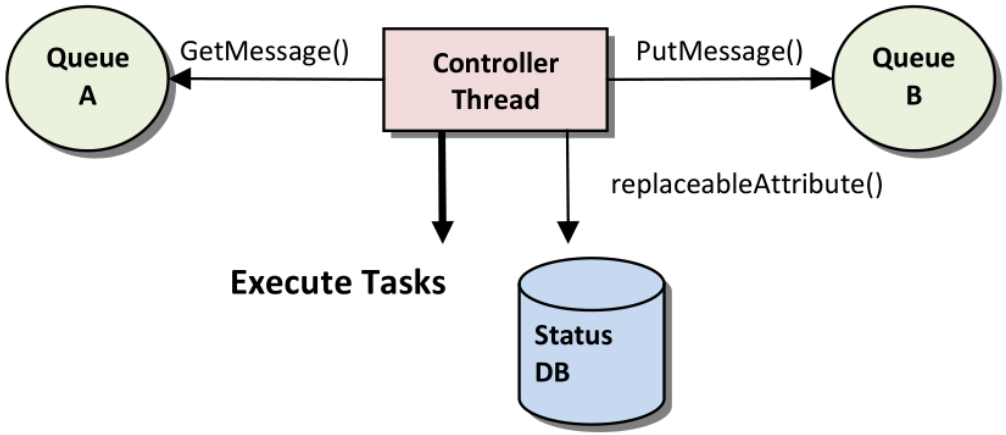


Loose coupling (independent phases using queues)

Source: amazon.com

Simple Controller Architecture for Resilience

- After designing the basic functionality, ask the question *"What if this fails?"* for **resilience**
- Here
  - Controllers externalize their state and thus can be resumed after failure
  - Visibility timeout in message queues: Messages become visible for other consumers if original consumer fails



1. Controller dequeues message from Queue A
2. Controller executes Tasks (for eg. Launch, monitor etc)
3. Controller Updates Statuses in status DB
4. Controller enqueues new message in Queue B

```
public abstract BaseController (SQSMessageQueue fromQueue, SQSMessageQueue
toQueue, SDBDomain domain)
```

Source: amazon.com

MapReduce

- **Definition:** MapReduce is a programming model and an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster.
- **Key Components:**
  - **Map Phase:** Processes input data into key-value pairs.
  - **Reduce Phase:** Aggregates values by key to produce the final output.
- **Purpose:** Facilitates scalable and fault-tolerant data processing.

- **Logical View of MapReduce**
  - **Map Function:** Transforms input key-value pairs into a list of intermediate key-value pairs.
    Syntax: Map(k1, v1) → list(k2, v2)
  - **Shuffle and Sort:** Groups all intermediate values by their key.
  - **Reduce Function:** Processes each group of sorted key-value pairs to produce a list of output key-value pairs.
    Syntax: Reduce(k2, list(v2)) → list(k3, v3)

Reference: https://en.wikipedia.org/wiki/MapReduce

MapReduce: Example and Shuffle and Sort Phase

- **Input Documents:**
  - "the quick brown fox”
  - "quick brown fox jumps over the lazy dog”
  - "the quick brown dog jumps over the fox”
- **Map Output:**
  - Emit each word with a count of 1.
  - *Example:* ("the", 1), ("quick", 1), ("brown", 1), etc.

- **Shuffle and Sort Phase**
  - **Process:**
    - Group all intermediate values by their key.
    - *Example:* All ("the", 1) pairs are grouped together.
  - **Purpose:** Ensures that all occurrences of the same word are sent to the same reducer.

Reference: https://en.wikipedia.org/wiki/MapReduce

MapReduce: Reduce Phase and Roles

- **Reduce Phase**
  - **Function:** Sums the counts for each word.
    - *Example:* For the key "the", the reducer receives a list of values [1, 1] and outputs ("the", 2).
  - **Output:** Final word counts.
    - *Example:* ("the", 2), ("quick", 3), ("brown", 3), etc.

- **Role of Map and Reduce**
  - **Map Phase:**
    - Processes and transforms input data into intermediate key-value pairs.
    - Enables parallel processing of input data.
  - **Reduce Phase:**
    - Aggregates intermediate data to produce final results.
    - Ensures that all values for a given key are processed together.

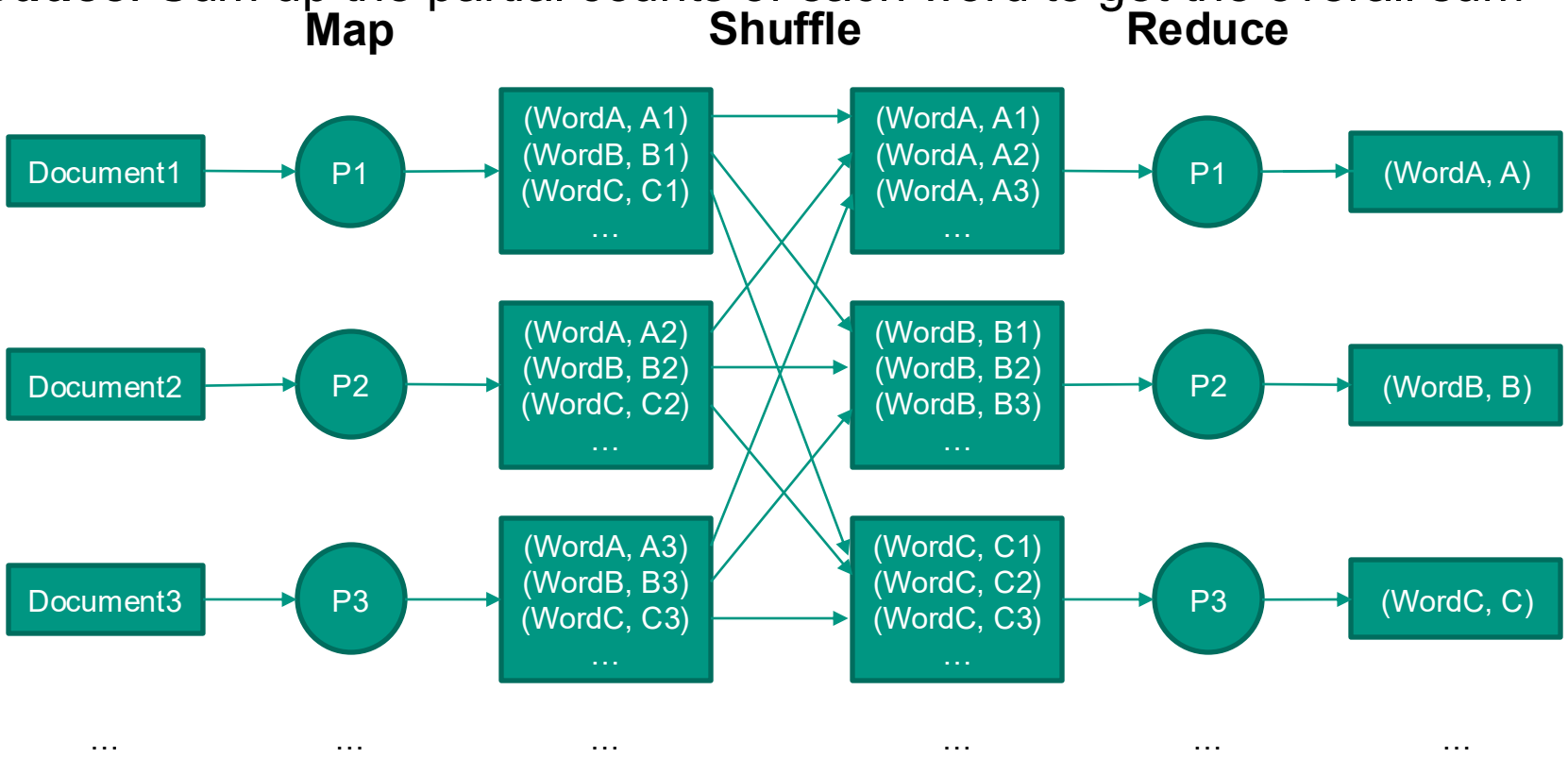
Reference: https://en.wikipedia.org/wiki/MapReduce

Example: Word Counts in Document Set

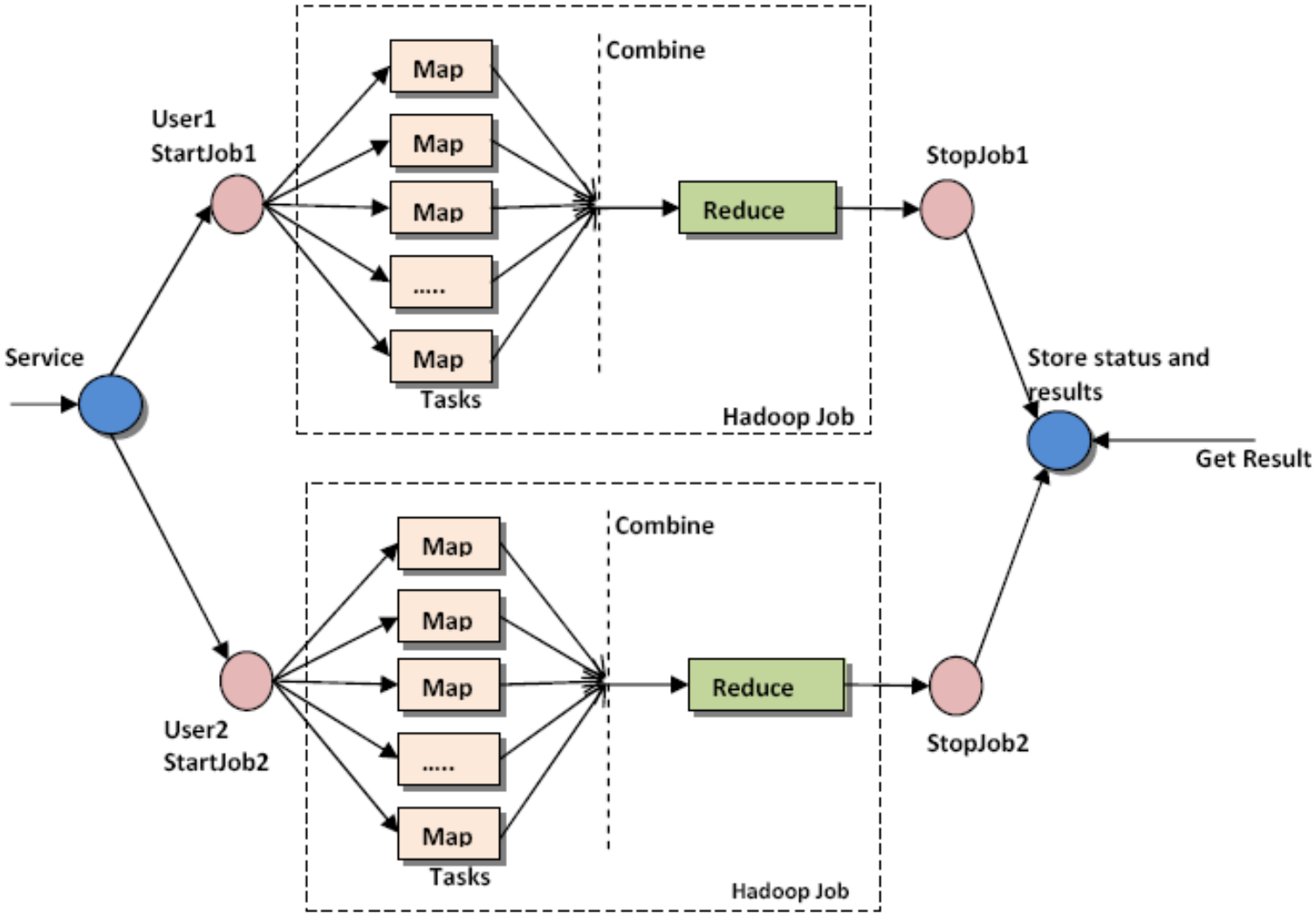
Map: Calculate count of each word for one document on each node

Shuffle (Combine): Move all counts for one word to the same computing node

Reduce: Sum up the partial counts of each word to get the overall sum



MapReduce



com

Guidelines for Cloud Applications

- Ensure that your application is scalable by designing **each component to be scalable** on its own.
- For better manageability and high-availability, make sure that your components are **loosely coupled**
- Implement **parallelization** for better use of the infrastructure and for performance
- After designing the basic functionality, ask the question *"What if this fails?"* for **resilience**
- Don't forget the **cost factor**. The key to building a cost-effective application is using on-demand resources in your design.

from [AmazonGrepTheWeb]

Architecture Principles (1)

- Decentralization:
  Use fully decentralized techniques to remove scaling bottlenecks and single points of failure.
- Asynchrony:
  The system makes progress under all circumstances.
- Autonomy:
  The system is designed, such that individual components can make decisions based on local information.
- Local responsibility:
  Each individual component is responsible for achieving its consistency; this is never the burden of its peers.

From http://s3.amazonaws.com/doc/s3-developer-guide/OverviewDesignPrinciple.html

Architecture Principles (2)

- Controlled concurrency:
  Operations designed requiring no or limited concurrency control.
- Failure tolerant:
  The system considers the failure of components to be a normal mode of operation, and continues operation with no or minimal interruption.
- Controlled parallelism:
  Abstractions used in the system are of such granularity that parallelism can be used to improve performance and robustness of recovery or the introduction of new nodes.

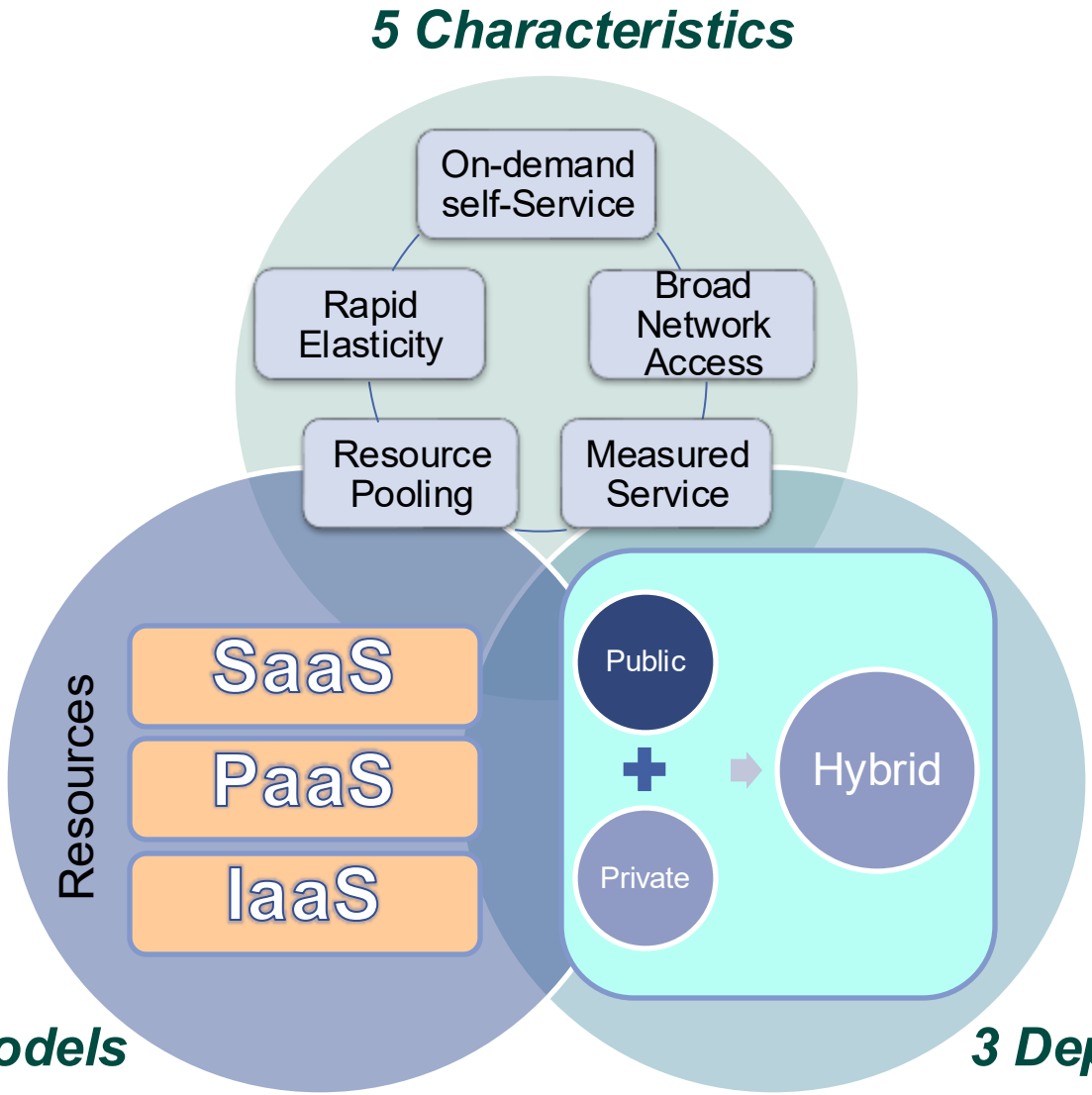
From http://s3.amazonaws.com/doc/s3-developer-guide/OverviewDesignPrinciple.html

Architecture Principles (3)

- Decompose into small well-understood building blocks:
  Do not try to provide a single service that does everything for everyone, but instead build small components that can be used as building blocks for other services.
- Symmetry:
  Nodes in the system are identical in terms of functionality, and require no or minimal node-specific configuration to function.
- Simplicity:
  The system should be made as simple as possible (but no simpler).

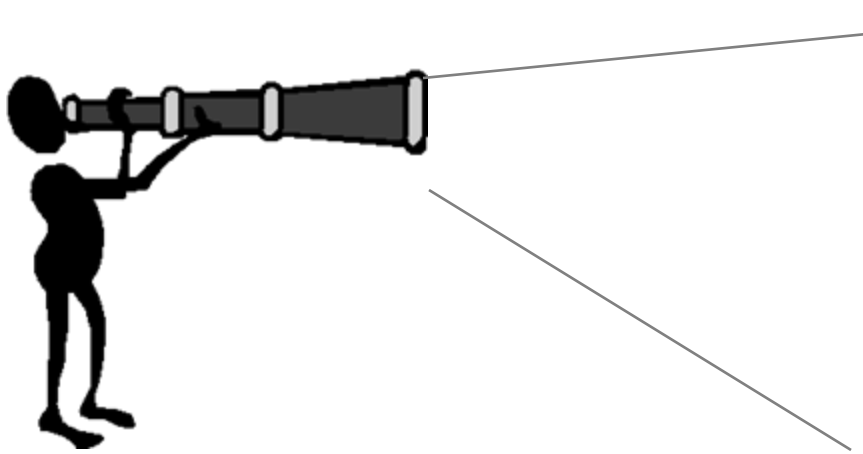
From http://s3.amazonaws.com/doc/s3-developer-guide/OverviewDesignPrinciple.html

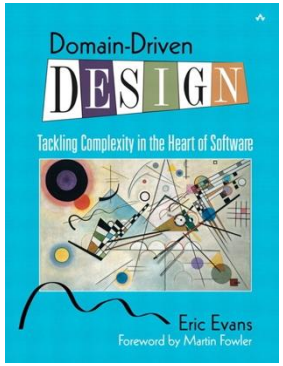
Conclusion: Cloud Computing Conceptional View [NIST]



Conclusion (continued)

- Shift from performance to elasticity
- Virtualisation techniques
- Multi-tenancy
- Map-Reduce-Programming Model
- Architectural principles: decentralisation, asynchrony, local responsibility, fault-tolerance, scalability, …





[Evans 2004]

APPENDIX

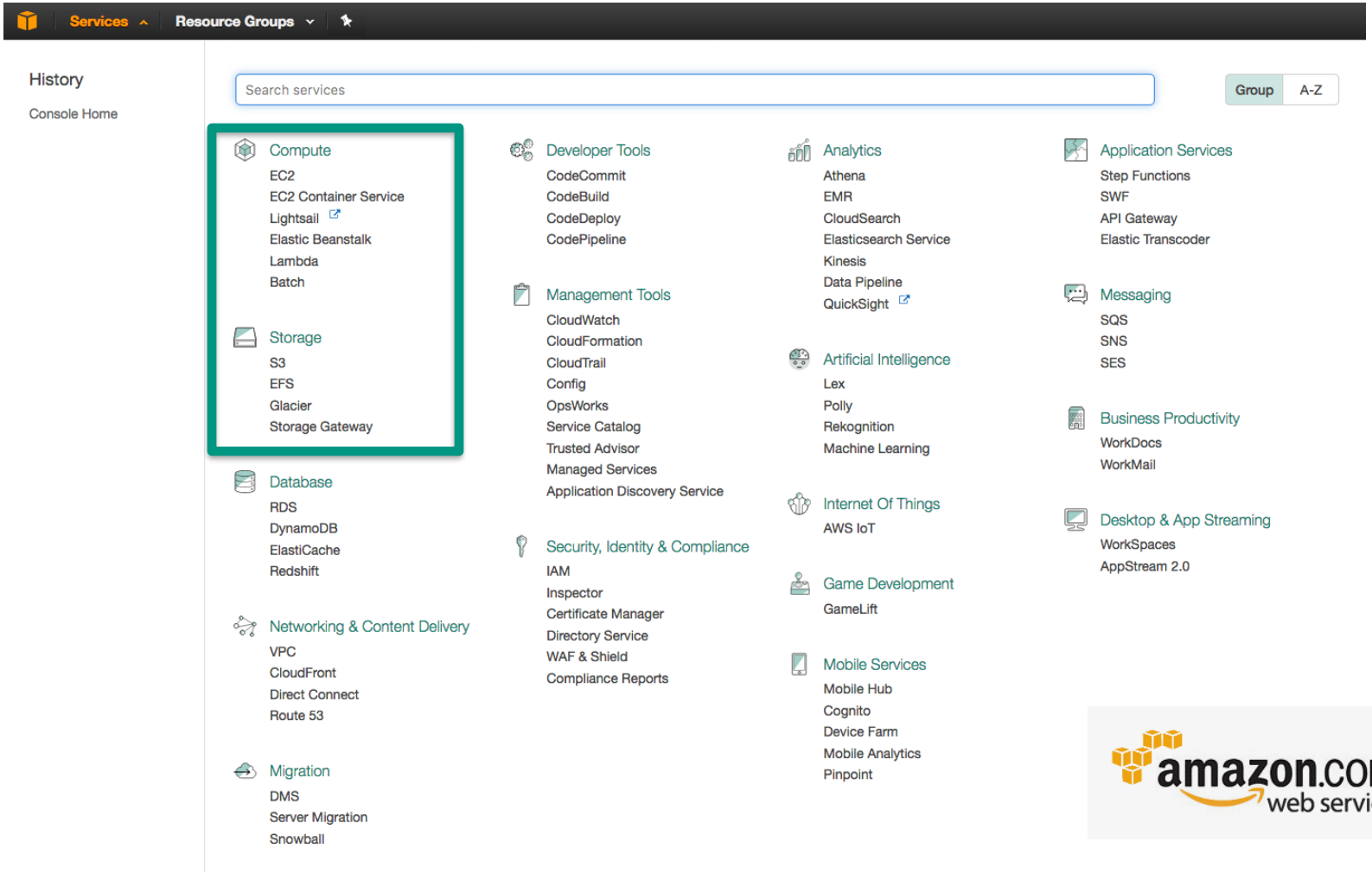
Examples

IaaS, PaaS, and SaaS

IaaS

Amazon Web Services (AWS)

http://aws.amazon.com/



Amazon Elastic Compute Cloud (EC2)

- Web-Service provides resizable compute capacity in the cloud
- Easy access of web-scale cloud computing for developers and architects
- Allows to launch instances with many different operating systems
  - Customized application environment
  - Customized network access permissions
- Supports Amazon Elastic Block Store
  - Scalable persistent block storage volumes
  - Supports high-performance, snapshots, replicas

EC2 Core Concepts

- **Amazon Machine Image (AMI): an encrypted file stored in Amazon storage, containing all the information necessary to boot instances of a customer's software**
  - An AMI is like a bootable root disk, which can be pre-defined or user-built.
    - Public AMIs: Pre-configured, template AMIs
    - Private AMIs: User-built AMI containing private applications, libraries, data and associated configuration settings

- **Instance: The running system based on an AMI**
  - All instances based on the same AMI begin executing identically. An instance can be launched in very few minutes.
  - Any information on them is lost when the instances are terminated or if they fail.
  - Data can be made persistent by use of the Amazon storage services

Amazon Simple Storage Service (S3)

- Provides storage through web services (REST, SOAP, BitTorrent)
- Designed for eleven nine durability (99.999999999%) and 99.99% availability of objects over a given year.
- Write, read, and delete objects containing up to 1 terabyte of data each. The number of objects you can store is unlimited.
- Each object is stored in a bucket and retrieved via a unique, developer-assigned key.
- Can be used for
  - Primary data storage
  - Backup & Archiving
  - Content Storage & Distribution
  - Big Data analysis
  - Cloud-native Application Data
  - Disaster recovery

S3

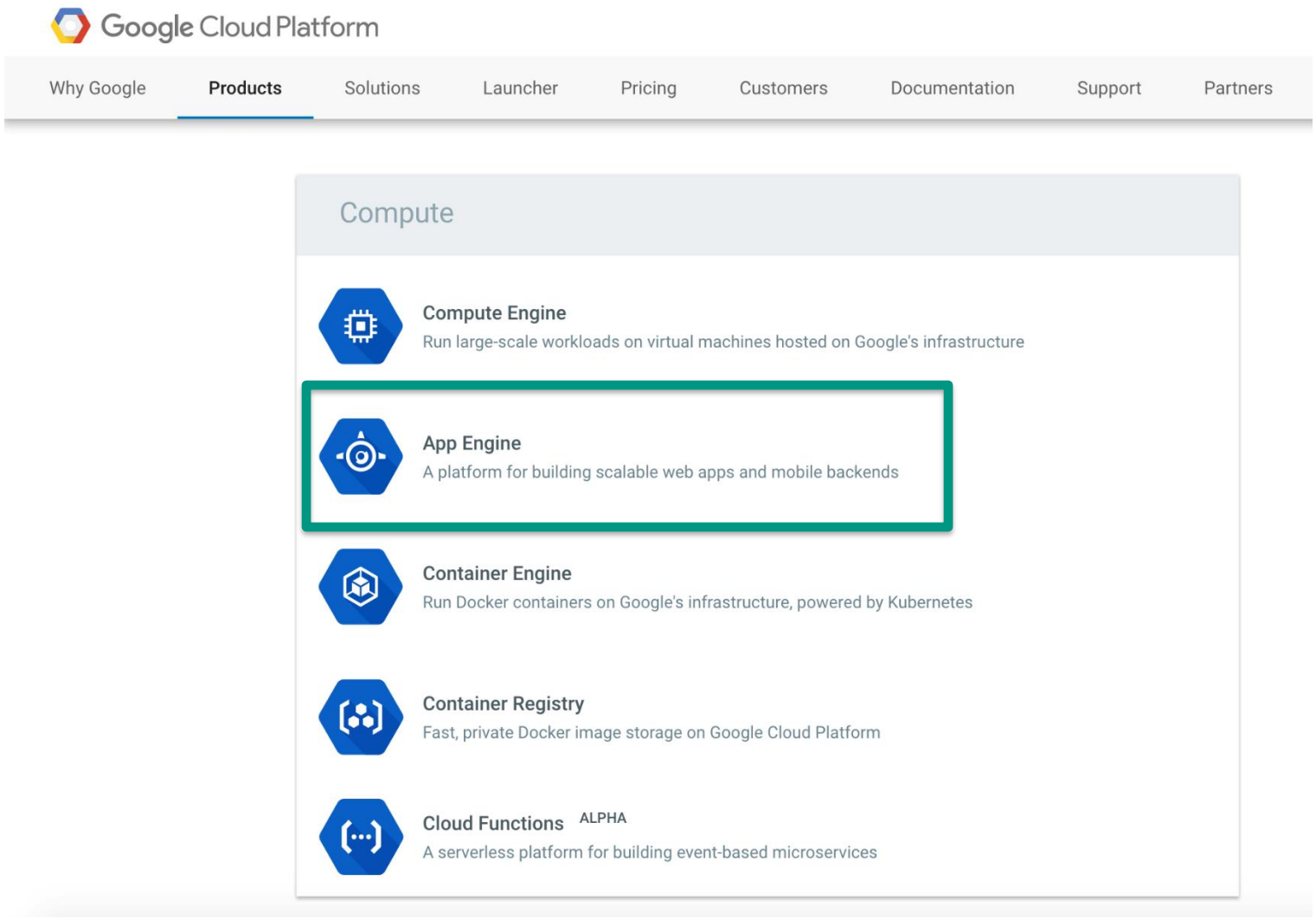
- A bucket can be located in the United States, in Europe or in Asia. All objects within the bucket will be stored in the bucket's location, but the objects can be accessed from anywhere.
- Authentication mechanisms are provided to ensure that data is kept secure from unauthorized access. Objects can be made private or public, and rights can be granted to specific users.
- Uses standards-based REST and SOAP interfaces designed to work with any Internet-development toolkit.
- Built to be flexible so that protocol or functional layers can easily be added. Default download protocol is HTTP. A BitTorrent™ protocol interface is provided to lower costs for high-scale distribution. Additional interfaces will be added in the future.
- Reliability backed with the Amazon S3 Service Level Agreement.

Source: aws.amazon.com

PaaS

Google Cloud Platform

https://cloud.google.com/products/

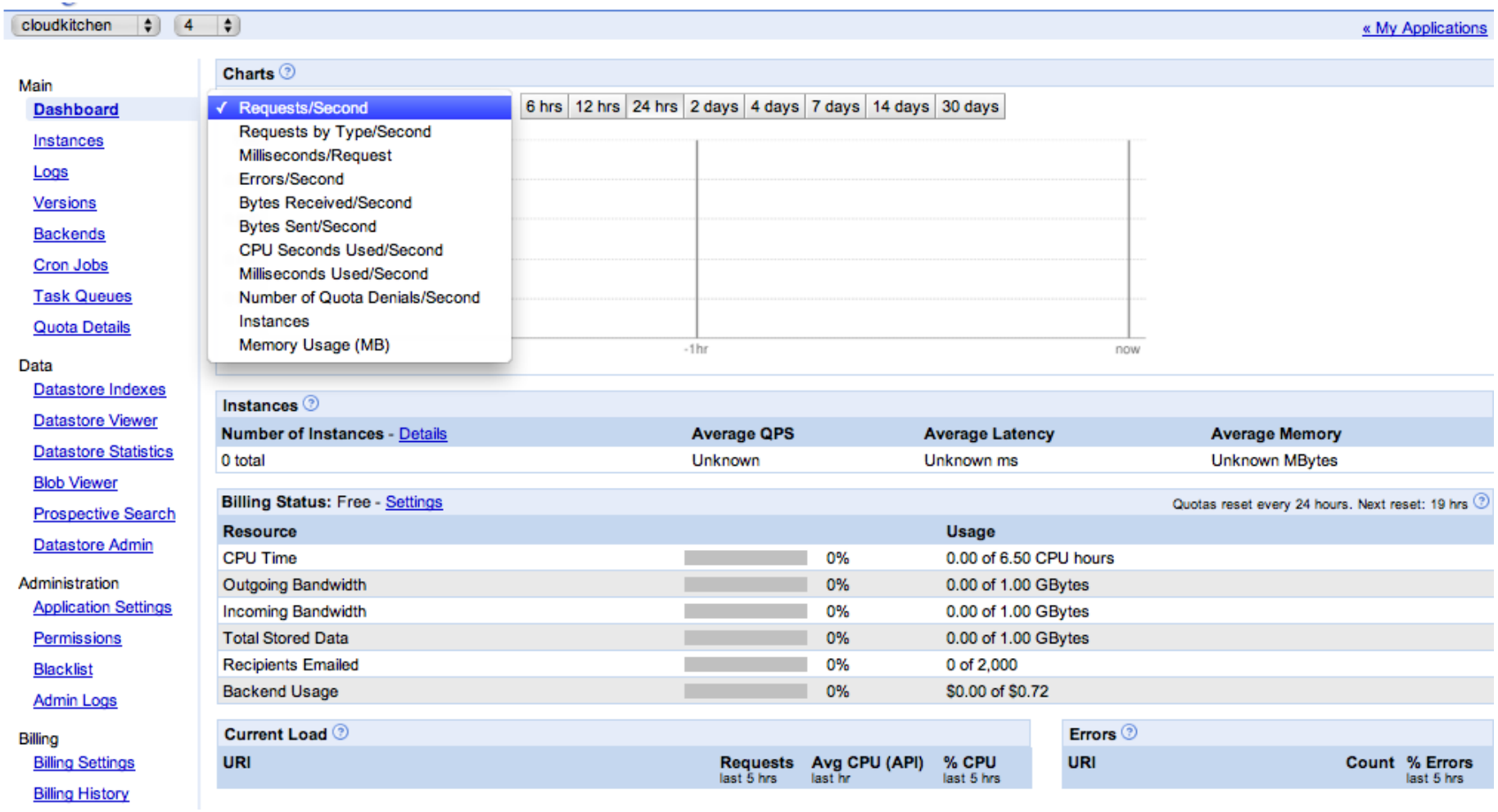


Google App Engine

- Platform as a Service cloud computing platform
- Used for developing and hosting of web applications
- Automatic scaling and load balancing
  - Vertical scalability = more resources in one instance
  - Horizontal scalability = more instances
- Many programming languages supported (Java, Python, PHP, GO, NodeJS,...)
- Sandbox support for enhancing security

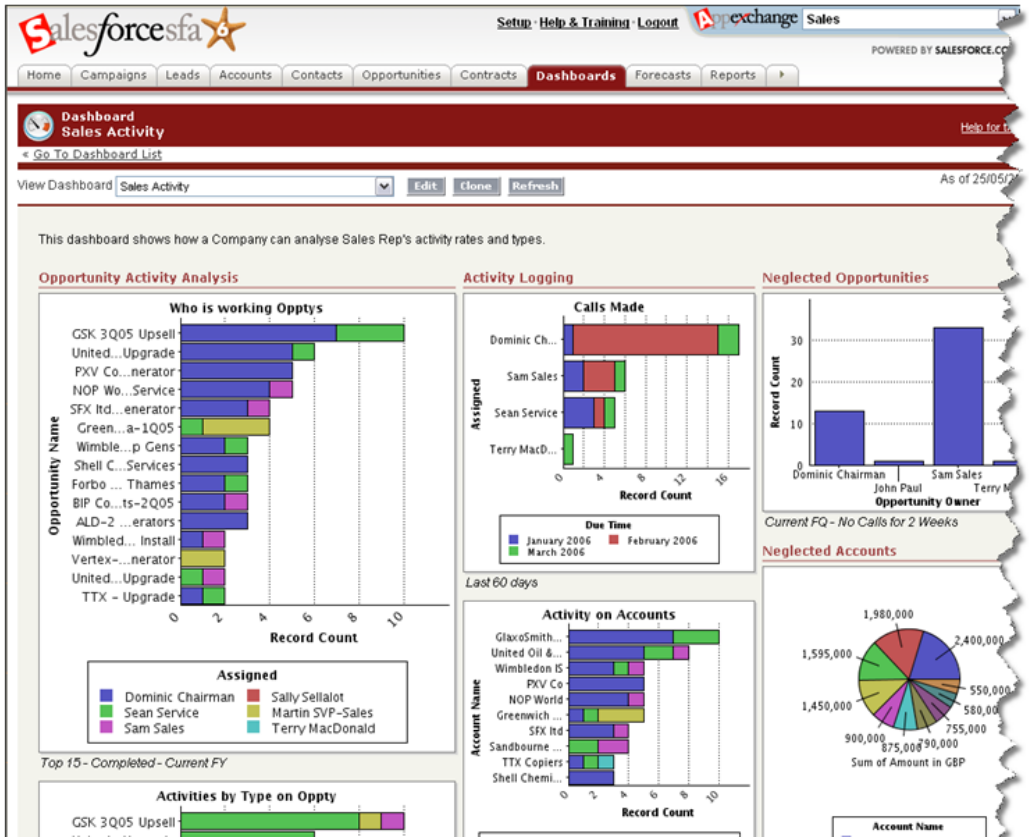
- However, several restrictions
  - File-system read only
  - Applications cannot fork new threads
  - 60 seconds time limit for request/response

Web-based Dashboard



SaaS

…started out primarily offering software for sales groups to manage their customer relationships

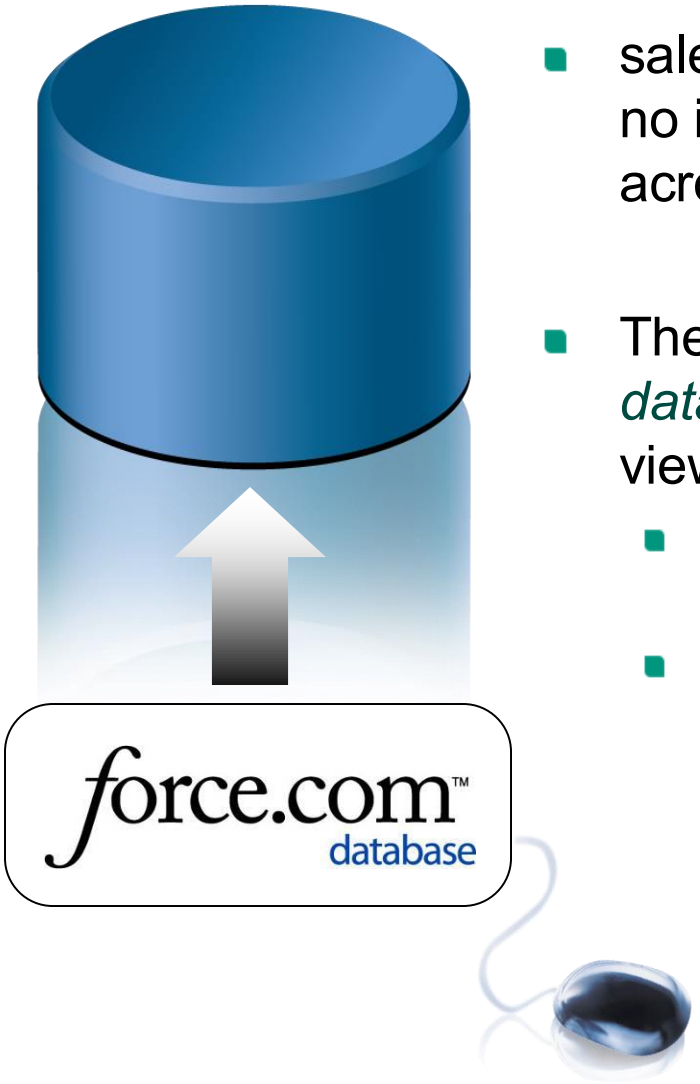


This included tools like address books, services for report generation with charts and graphs, and applications in support of more complicated processes such as tracking potential customers from lead to sale

Infrastructure is strategic asset

- Salesforce.com includes
  - built-in customer base
  - built-in distribution through the force.com appexchange directory
  - built-in data and authentication models
  - developer support on the appexchange developer network, and
  - several programming tools
- This infrastructure has been flexible enough to allow salesforce to branch out into other business applications like marketing and customer support
- It has also been flexible enough to allow customers to build their own applications in areas like financial services and human resources

salesforce.com & force.com



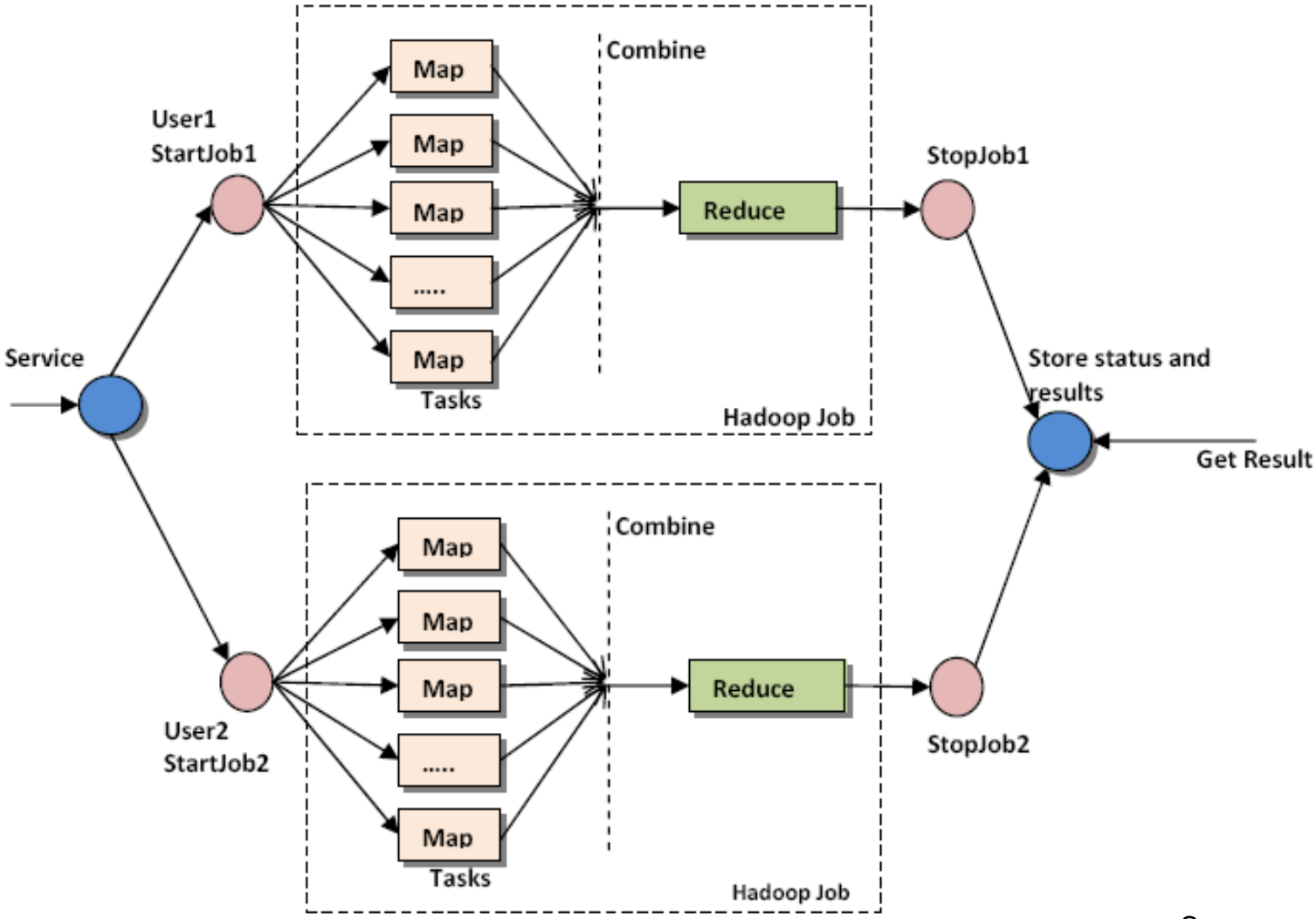
- salesforce.com is entirely on demand: there is no installed software, but the platform runs across the internet in a SaaS model
- The *underlying technology is based around database concepts*, with default actions and views
  - Salesforce has almost completely opened up this infrastructure: force.com
  - Even novice users can create custom objects that are the equivalent of database tables, and add or remove fields from the default objects

Steve Bobrowski: The Force.com Multitenant Architecture. Understanding the Design of Salesforce.com's Internet Application Development Platform, available online: http://wiki.developerforce.com/index.php/Multi_Tenant_Architecture

Main Idea in a Nutshell

- Separate a compiled runtime environment (“kernel”) from several (meta-)data layers (data, common metadata and tenant-specific metadata)
- When a user creates a custom extension, the extension is saved in a metadata directory and created on runtime (thus improving scalability)
- A potential bottleneck are metadata I/O operations which is why caching of the “virtual applications” + directory search optimization is a good idea
- Force.com provides developers with a WSDL document that lets them generate an API for accessing the Force.com Web services

MapReduce



Source: amazon.com

Google's MapReduce Programming Model

- Input & Output: each a set of key/value pairs
- Programmer specifies two functions:
- `map (in_key, in_value) -> list(out_key, intermediate_value)`
  - Processes input key/value pair
  - Produces set of intermediate pairs
- `reduce (out_key, list(intermediate_value)) -> list(out_value)`
  - Combines all intermediate values for a particular key
  - Produces a set of merged output values (usually just one)
- Inspired by similar primitives in LISP and other languages

```
function map(String name, String document):
// name: document name
// document: document contents
for each word w in document:
  emit (w, 1)

function reduce(String word, Iterator partialCounts):
// word: a word
// partialCounts: a list of aggregated partial counts
sum = 0
for each pc in partialCounts:
  sum += pc
emit (word, sum)
```

From http://research.google.com/archive/mapreduce-osdi04-slides/index-auto-0003.html and https://en.wikipedia.org/wiki/MapReduce