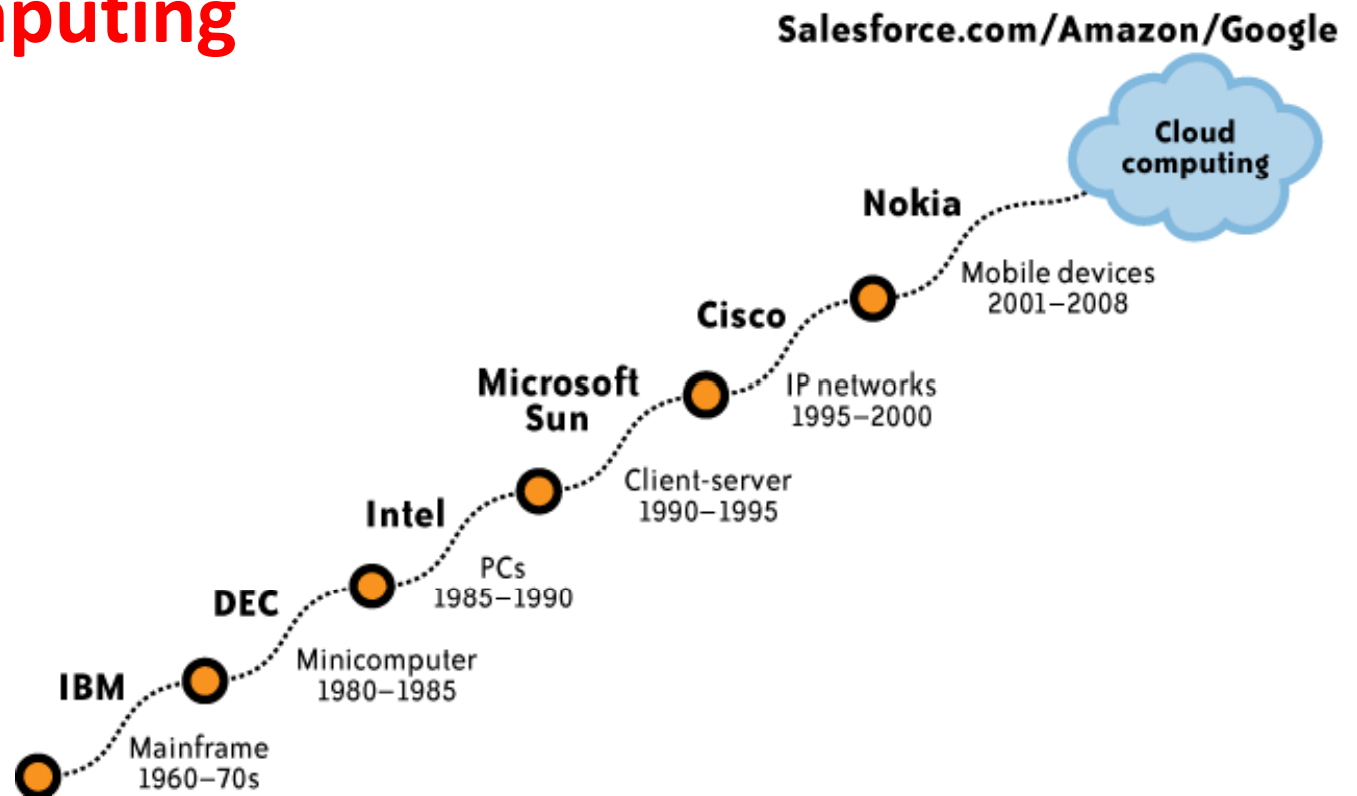


# Cloud Computing Architecture

# Definition of Cloud Computing

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

# Evolution of model computing



## What is Cloud Computing?

**C**ommon,  
**L**ocation-independent,  
**O**nline  
**U**tility that is available on  
**D**emand

--- (Chan, 2009)

# Essential Cloud Characteristics

- On-demand self-service
  - Get computing capabilities as needed automatically
- Broad network access
  - Services available over the net using desktop, laptop, PDA, mobile phone

## Essential Cloud Characteristics (Cont.)

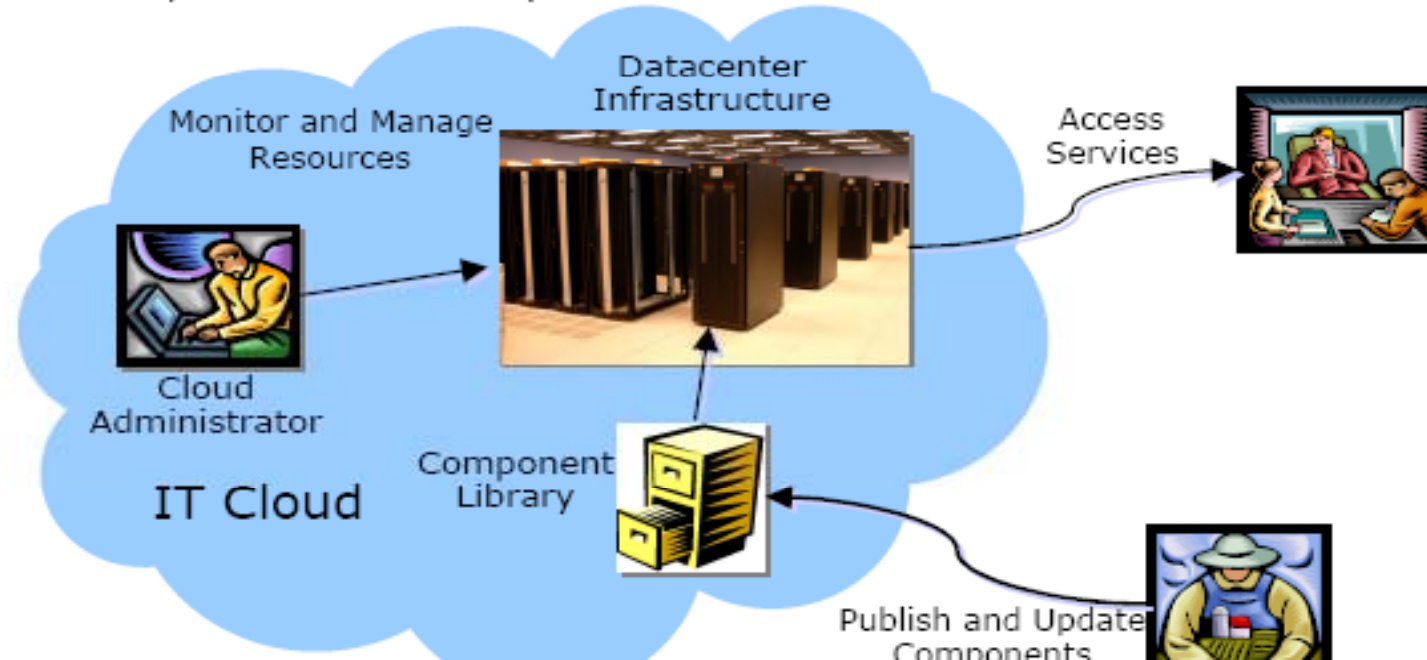
- Resource pooling
  - Location independence
  - Provider resources pooled to server multiple clients
- Rapid elasticity
  - Ability to quickly scale in/out service
- Measured service
  - control, optimize services based on metering

# Common Cloud Characteristics

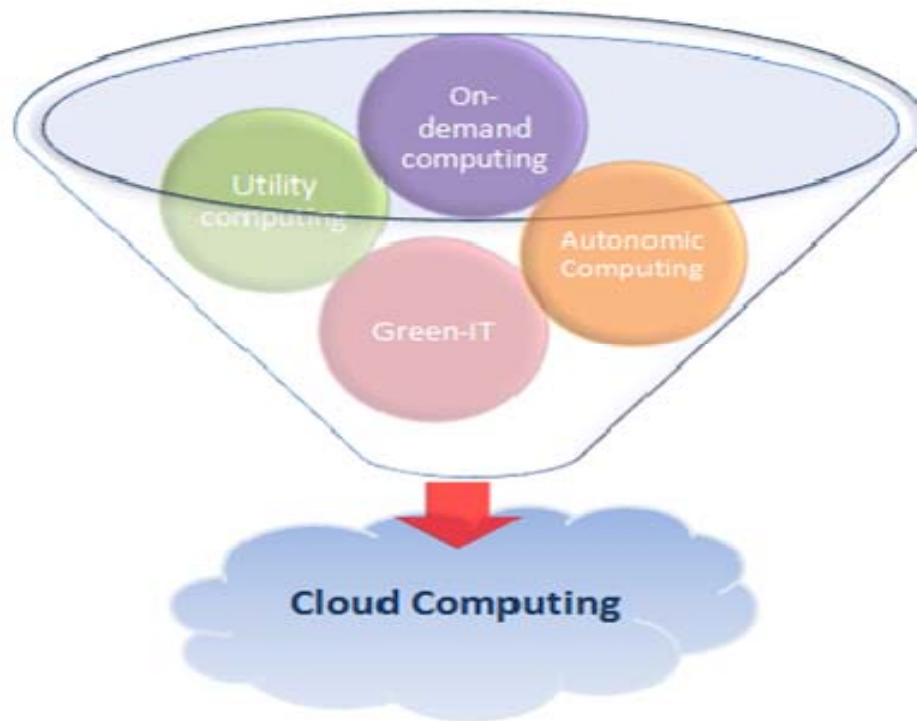
- Cloud computing often leverages:
  - Massive scale
  - Homogeneity
  - Virtualization
  - Resilient computing
  - Low cost software
  - Geographic distribution
  - Service orientation
  - Advanced security technologies

Cloud Computing is an emerging style of computing in which **applications**, **data**, and **resources** are **provided as services** to users over the Web

- Services provided may be available globally, always on, low in cost, “on demand”, massively scalable, “pay as you grow”, ...
- Consumers of the services need only care about *what* the service does for them, *not how* it is implemented

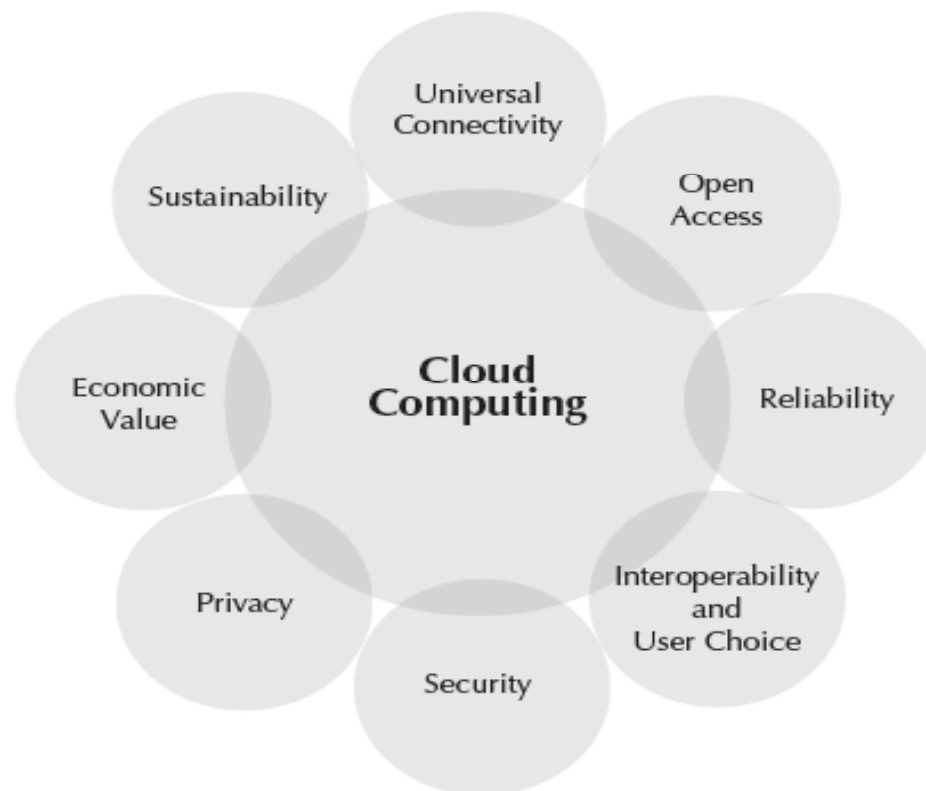






Cloud computing is an emerging business model that delivers computing services over the Internet in an elastic self-serviced, self-managed, cost-effective manner with guaranteed Quality of Service(QoS).

## Fundamental Elements of cloud Computing



## Five characteristics of cloud computing

Resource-Pooling  
On-Demand-Self-Service  
Scalable  
Uses-Internet-Technologies  
Metered-by-Use  
Elastic

[www.techno-pulse.com](http://www.techno-pulse.com)

## Pros and Cons of cloud computing



## **Types of clouds (Cloud Deployment Models)**

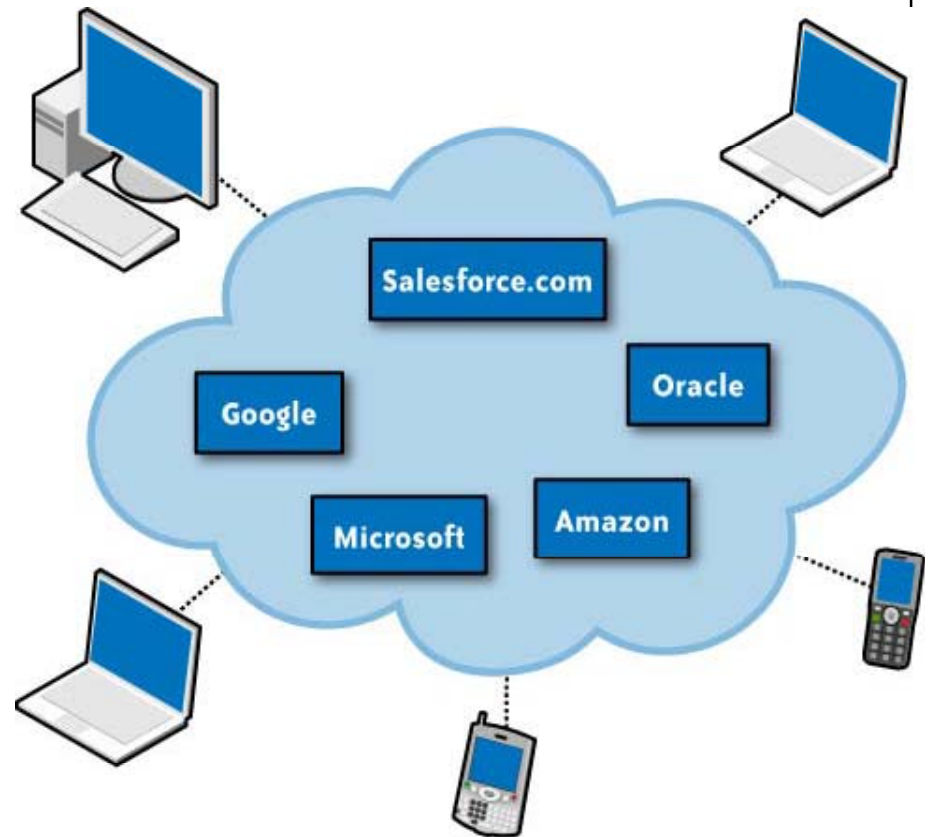
Public/External cloud

Hybrid/ Integrated cloud

Private/Internal cloud

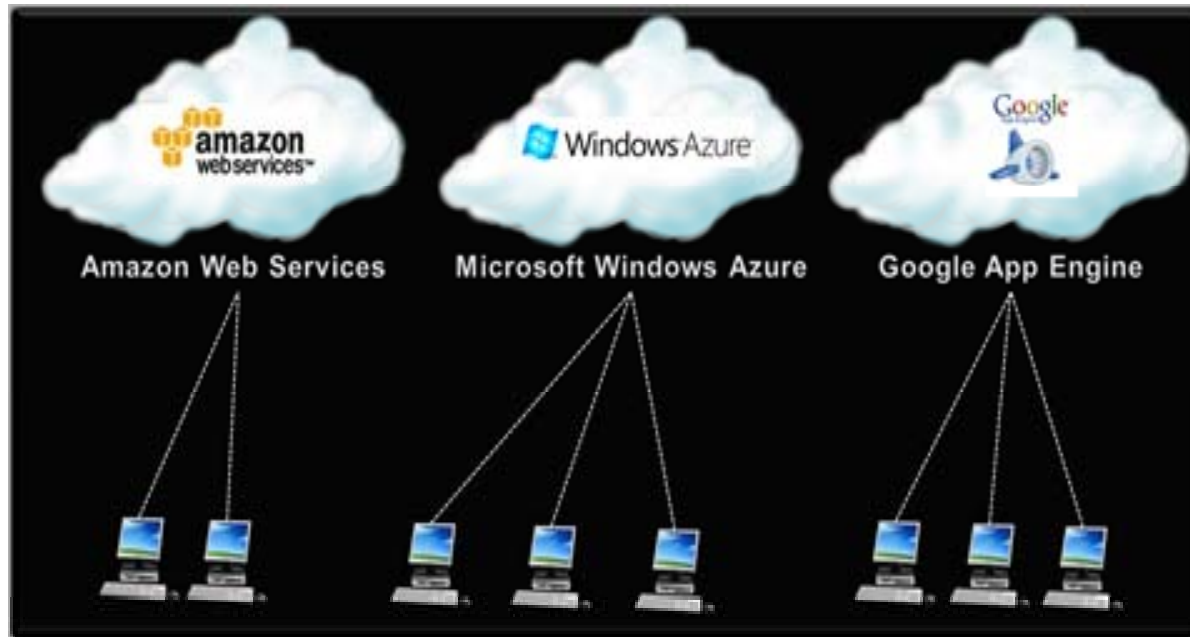
Community/Vertical Clouds

A **public cloud** (also called External Cloud) is one based on the standard cloud computing model, in which a service provider makes resources, such as applications and storage, available to the general public over the Internet. Public cloud services may be free or offered on a pay-per-usage model.



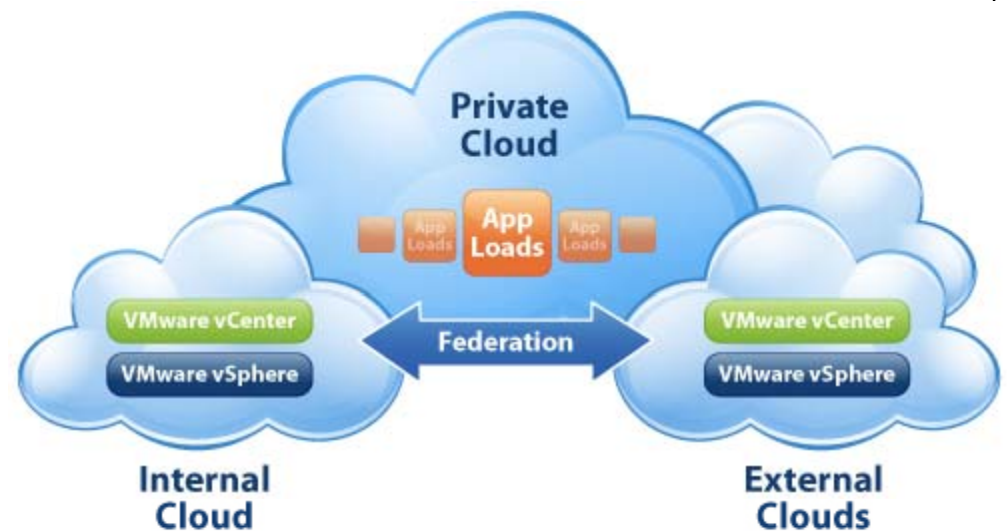
## The main benefits of using a public cloud service are:

- Easy and inexpensive set-up because hardware, application and bandwidth costs are covered by the provider.
- Scalability to meet needs.
- No wasted resources because you pay for what you use.



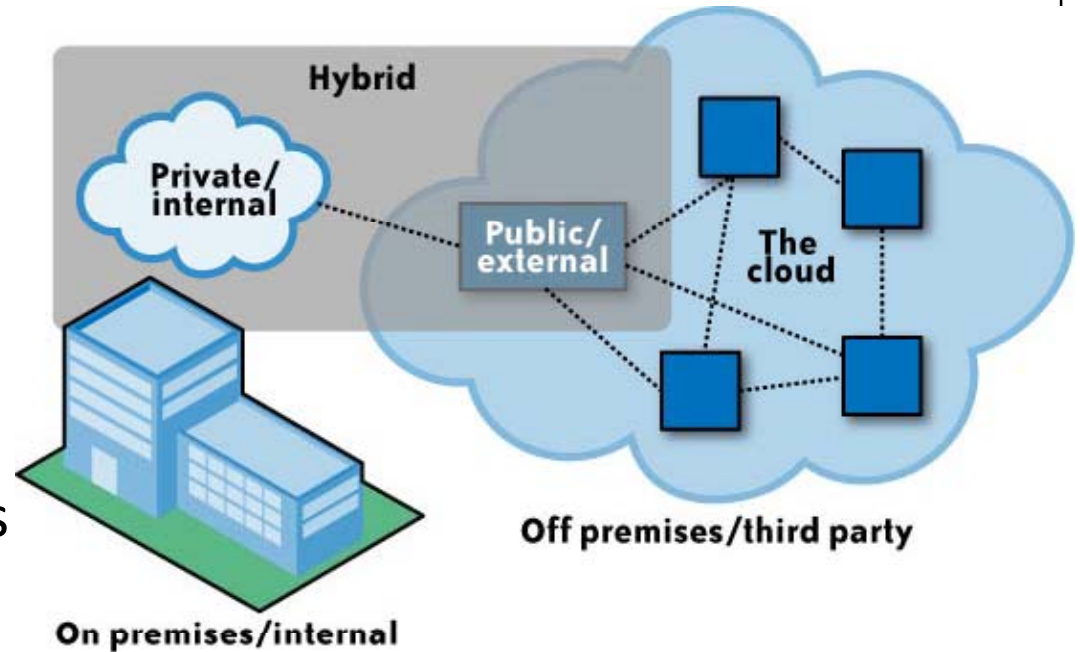
**Private cloud** (also called internal cloud or corporate cloud) is a marketing term for a proprietary computing architecture that provides hosted services to a limited number of people behind a firewall.

→ Advances in virtualization and distributed computing have allowed corporate network and datacenter administrators to effectively become service providers that meet the needs of their "customers" within the corporation.





**A hybrid cloud** is a composition of at least one private cloud and at least one public cloud. A hybrid cloud is typically offered in one of two ways: a vendor has a private cloud and forms a partnership with a public cloud provider, or a public cloud provider forms a partnership with a vendor that provides private cloud platforms



**Community clouds** are a deployment pattern suggested by **NIST**, where semi-private clouds will be formed to meet the needs of a set of related stakeholders or constituents that have common requirements or interests.

→ Communities of Interest (COI) constructs typical of the federal government may be enabled by community clouds to augment their wiki-centric collaboration processes with cloud enabled capabilities as well.

# Cloud reference model

## Cloud Service Models :

Cloud Computing can be broadly classified into three **\*aaS**, i.e., three layers of Cloud Stack, also known as **Cloud Service Models** or **SPI Service Model**:

- Infrastructure-as-a-Service (**IaaS**)
- Platform-as-a-Service (**PaaS**)
- Software-as-a-Service (**SaaS**)

# Layers

- Client
- Application
- Platform
- Infrastructure
- Server

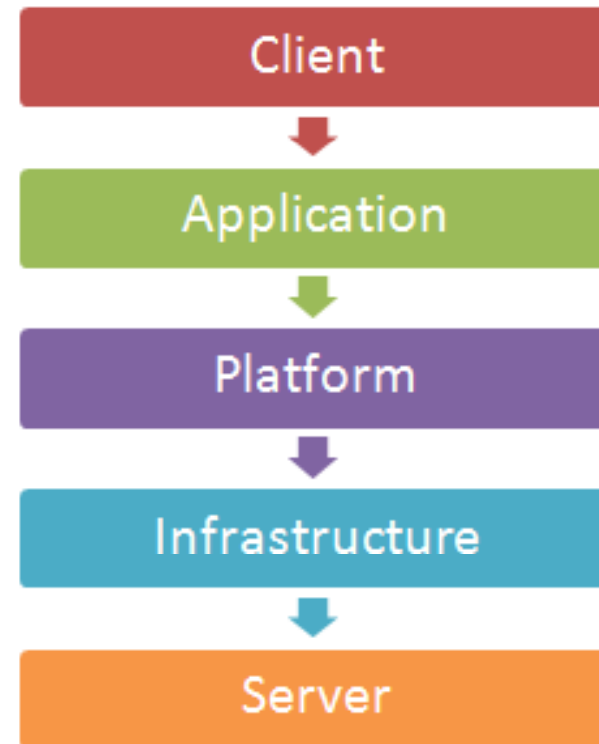
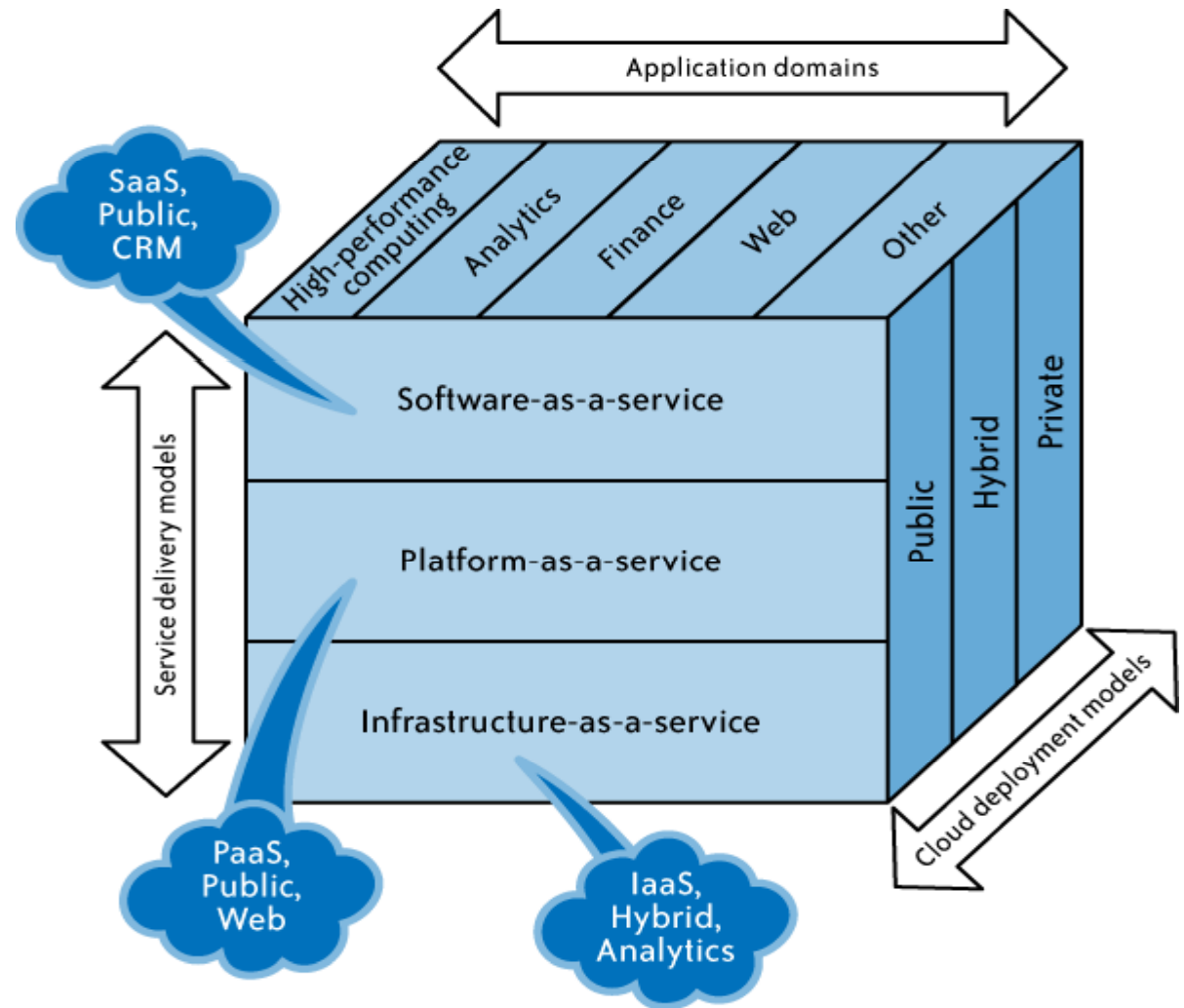
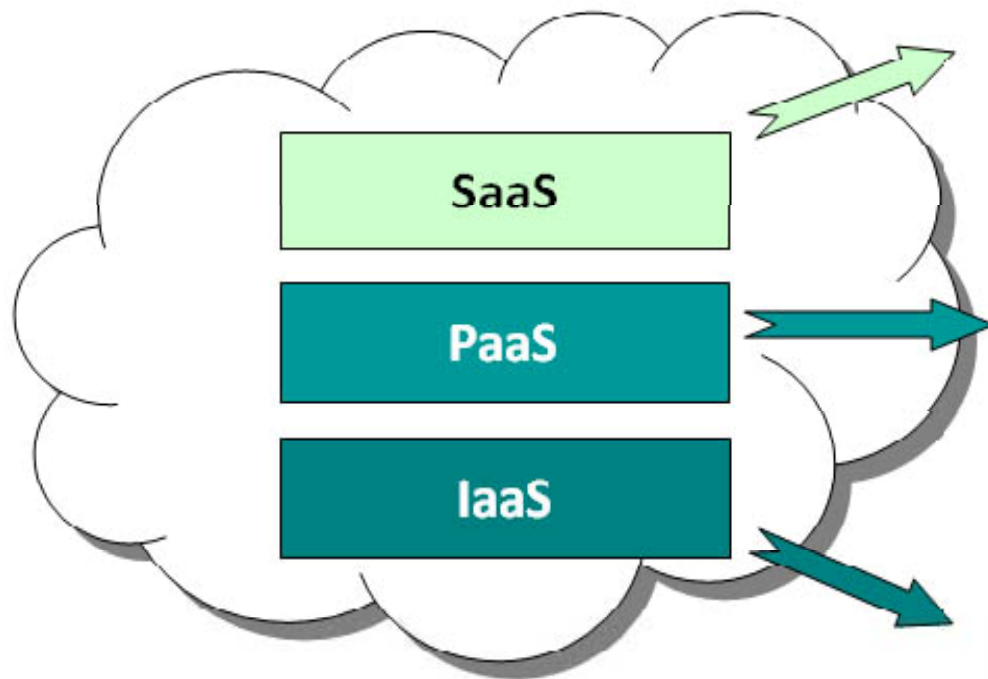







Diagram for :  
Relationship  
between  
services, uses,  
and types of  
clouds





Who Uses It	What Services are available	Why use it?
Business Users	EMail, Office Automation, CRM, Website Testing, Wiki, Blog, Virtual Desktop ...	To complete business tasks
Developers and Deployers	Service and application test, development, integration and deployment	Create or deploy applications and services for users
System Managers	Virtual machines, operating systems, message queues, networks, storage, CPU, memory, backup services	Create platforms for service and application test, development, integration and deployment

	Amazon	Google	Salesforce	Customer Implications
Software as Service				<ul style="list-style-type: none"> <li>+ Application logic, platform and infrastructure abstracted</li> <li>+ Significant reduction in effort to deploy, run and manage</li> <li>- Apps can be configured but may not meet highly customized requirements</li> </ul>
Platform as Service				<ul style="list-style-type: none"> <li>+ Platform &amp; infrastructure abstracted</li> <li>+ Custom apps can be built order of magnitude more quickly and cheaply</li> <li>- Custom apps still need to be supported and managed</li> </ul>
Infrastructure as Service				<ul style="list-style-type: none"> <li>+ Physical infrastructure abstracted</li> <li>+ Can be scaled up and down as needed</li> <li>- Needs to be provisioned/managed</li> <li>- Higher levels of stack still need to be managed, maintained and supported</li> </ul>



SaaS  
Software as a Service

PaaS  
Platform as a Service

IaaS  
Infrastructure as a Service

SaaS  
Software as a Service

A blue rounded square icon with a black border, containing the text 'SaaS' in black serif font.

SaaS

# Software delivery model

- Increasingly popular with SMEs
- No hardware or software to manage
- Service delivered through a browser

A blue rounded square icon with a black border, containing the text 'SaaS' in black.

SaaS

# Advantages

- Pay per use
- Instant Scalability
- Security
- Reliability
- APIs (application programming Interface)

SaaS

## Examples

- CRM
- Financial Planning
- Human Resources
- Word processing

## Commercial Services:

- Salesforce.com
- emailcloud

PaaS  
Platform as a Service

# Platform delivery model



PaaS

- Platforms are built upon Infrastructure, which is expensive
- Estimating demand is not a science!
- Platform management is not fun!

# Popular services



PaaS

- Storage
- Database
- Scalability



# Advantages



PaaS

- Pay per use
- Instant Scalability
- Security
- Reliability
- APIs

# Examples

PaaS

- Google App Engine
- Mosso
- AWS: S3

IaaS

Infrastructure as a Service

# Computer infrastructure delivery model

Access to infrastructure stack:

- Full OS access
- Firewalls
- Routers
- Load balancing



IaaS

# Advantages

- Pay per use
- Instant Scalability
- Security
- Reliability
- APIs

A yellow rounded square box with a black border, containing the text 'IaaS'.

IaaS

# Examples

- Flexiscale
- AWS: EC2

A yellow rounded square icon with a black border, containing the text 'IaaS' in black serif font.

IaaS

SaaS  
Software as a Service

PaaS  
Platform as a Service

IaaS  
Infrastructure as a Service

SaaS

PaaS

IaaS

# Common Factors

- Pay per use
- Instant Scalability
- Security
- Reliability
- APIs



SaaS

# Advantages

PaaS

- Lower cost of ownership
- Reduce infrastructure management responsibility
- Allow for unexpected resource loads
- Faster application rollout

IaaS

SaaS

PaaS

IaaS

# Cloud Economics

- Virtualisation lowers costs by increasing utilisation
- Economies of scale afforded by technology
- Automated update policy

SaaS

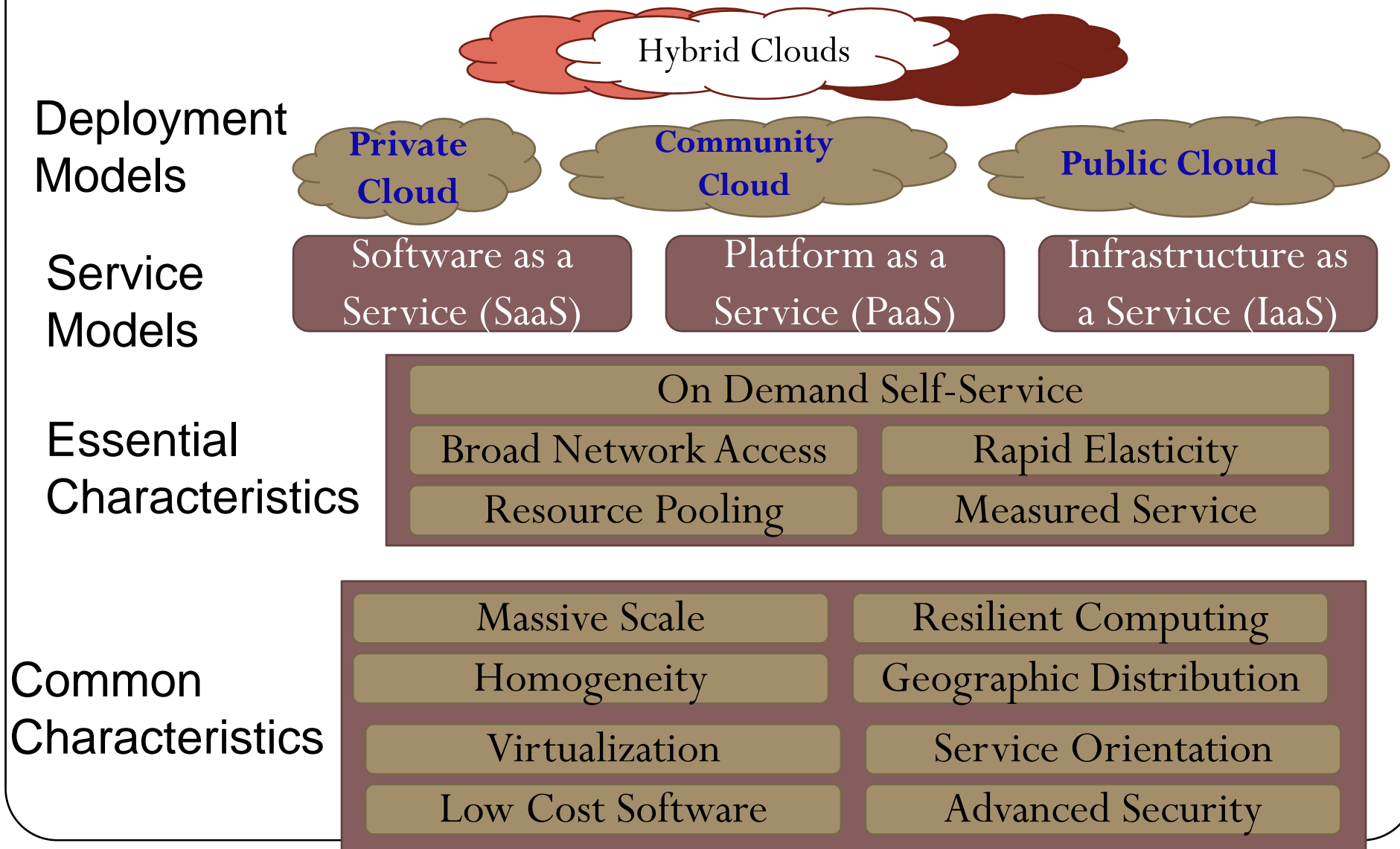
## Risks

- Security
- Downtime
- Access
- Dependency
- Interoperability

PaaS

IaaS

# The NIST (National Institute of Standards and Technology) Cloud Definition Framework



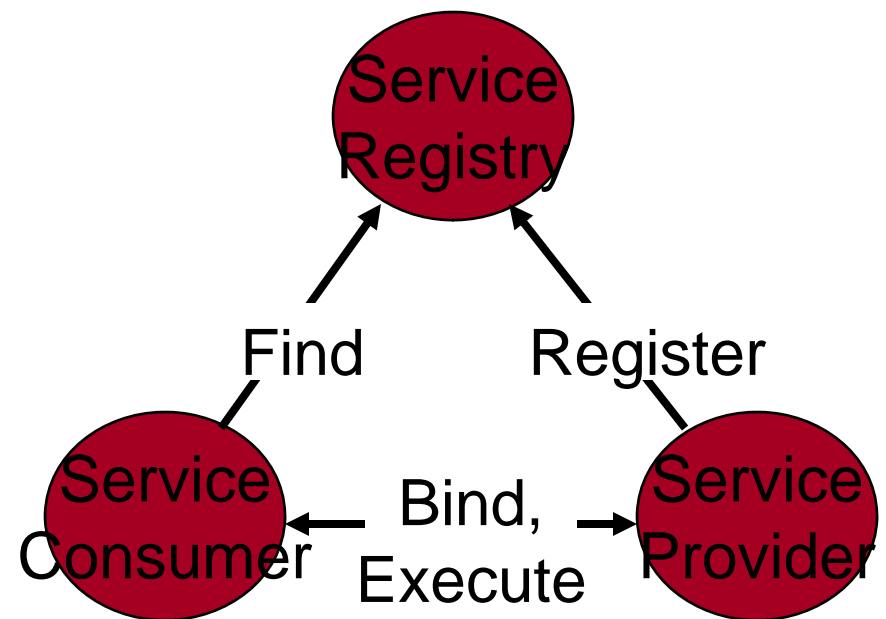
# **SERVICE ORIENTED ARCHITECTURE**

# What is Service Oriented Architecture (SOA)?

- “A service-oriented architecture is essentially a collection of services. These services communicate with each other.
- The communication can involve either simple data passing or it could involve two or more services coordinating some activity.
- Some means of connecting services to each other is needed.”
- “Service-oriented architecture (SOA) provides methods for systems development and integration where systems group functionality around business processes and package these as *interoperable services*.
- An SOA infrastructure allows different applications to exchange data with one another as they participate in business processes.
- SOA separates functions into distinct units, or services, which developers make accessible over a network in order that users can combine and reuse them in the production of business applications “

# What is Service Oriented Architecture (SOA)?

- Is not a computing architecture but a style of programming
- An SOA application is a composition of services
- A “service” is the building block/ unit of an SOA
- Services encapsulate a business process
- Service Providers Register themselves
- Service use involves: Find, Bind, Execute
- Most well-known instance is Web Services



# SOA Actors

- Service Provider

- From a business perspective, this is the owner of the service. From an architectural perspective, this is the platform that provides access to the *service*.

- Service Registry

- This is an information space of service *descriptions* where service providers publish their services and service requesters find services and obtain binding information for services.
- Allows service consumers to locate service providers that meet required criteria

- Service Consumer

- From a business perspective, this is the business that requires certain function to be fulfilled. From an architectural perspective, this is the *client* application that is looking for and eventually invoking a service.



# SOA Principles

- Formal contract
- Loose coupling
- Abstraction
- Reusability
- Autonomy
- Statelessness
- Discoverability
- Composability

Thomas Erl, SOA Principles of Service Design, Prentice Hall 2007 ISBN:0132344823

# SOA Principles – Formal contract

- According to SOA Formal contract principle every service needs to have an official, standardized, formal contract.
- A great deal of emphasis is placed on specific aspects of contract design, including:
  - the manner in which services express **functionality (functional description contract)**
  - how data types and data models are defined (**information model**)
  - how policies are asserted and attached. (**non-functional description contract**)
  - how interaction with the service is to be performed (**behavioral contract**)

# SOA Principles – Loose coupling

- SOA is a loosely coupled arrangement of services and service consumers. At design time, loose coupling means that services are designed with no affinity to any particular service consumer. Inside the service, no information is assumed as to the purpose, technical nature or business nature of the service consumer. Thus, a service is fully decoupled from a service consumer.
- However, the service consumer is dependent on the service (that is, it embeds literal references to service interfaces). Thus, SOA is asemi-coupled (or *loosely coupled*) architecture. It differs from an *event-driven architecture*, in which all participating software components are decoupled from others, and also from a *monolithic architecture*, in which all software components are designed to operate only in the initially intended context (that is, logically tightly coupled).
- Design-time loose coupling is essential to SOA because it enables the non-intrusive reuse of service interfaces. However, tools can't guarantee design-time loose coupling. *Poorly designed services, which are logically locked into their service consumers, may render the entire application monolithic — despite the use of SOA-style technologies.*

“Introduction to Service-Oriented Architecture”, YefimV. Natis, Roy W. Schulte,  
14 April 2003

# SOA Principles – Abstraction

- This principle emphasizes the need to hide as much of the underlying details of a service as possible.
- By using abstraction previously described loosely coupled relationship is directly enabled and preserved
  - There are 4 levels of abstraction in SOA as:
    - *technology abstraction*
    - *functional abstraction*
    - *programming logic abstraction*
    - *quality of service abstraction*

# SOA Principles – Reusability

- The reusability principle suggest to contain and express agnostic logic as services that can be positioned as reusable enterprise resources
- Reusability will:
  - Allow for service logic to be repeatedly leveraged over time so as to achieve a high Return on investment( ROI)
  - Increase business agility on an organizational level
  - Enable the creation of service inventories that can be easily integrated and used in various use-cases

Thomas Erl, SOA Principles of Service Design, Prentice Hall 2007 ISBN:0132344823

# SOA Principles – Autonomy

- SOA Autonomy principle implies that services have control over the solution logic they implement.
- SOA Autonomy/ Service Autonomy can be observed as various levels:
  - **Runtime autonomy** – represents the amount of control a service has over its execution environment at runtime
  - **Design-time autonomy** – represents the amount of governance control a service owner has over the service design

# SOA Principles – Statelessness

- This means a service must do its best to hold onto state information pertaining to an interaction for as small a duration as possible, e.g., do not retain awareness of a message once it is processed.
- Statelessness in a service means that if the service is enlisted in a flow, than it doesn't retain any state referring to the enclosing flow. Form a message perspective, it means that once a service has received and processed a message, it doesn't retain memory of the passage of that message.
- This helps with concurrent access scaling

# Statelessness in SOA and REST

- SOA and REST share the Statelessness principle
- REST provides explicit state transitions
- REST Servers are stateless and messages can be interpreted without examining history.
- Persistent data can be given explicit URIs on the server.
- Messages can refer to persistent data through links to Uniform Resource Identifier(URI)s.



# Statelessness in SOA and REST

- **In SOA**

- Stateless communication although communication can be stateful as well
  - Received or sent messages can trigger state change
  - Operations requiring sequence of messages
- Capable to support transactions
  - set of operations with pass or fail results
- Tighter coupling between components

- **In REST**

- Stateless communication
  - Document transfer only
  - A party is not aware of its partner current state
- Party receiving information can decide how to process it
- HTTP caching possible
- Looser coupling between components

# SOA Principles – Discoverability

- SOA Discoverability is meant to help one avoid the accidental creation of services that are either redundant or implement logic that is redundant. The discoverability principle can be referred to the ***design of an individual service so that it becomes as discoverable as possible*** – no matter whether the discoverability extension or product actually exists in the surrounding implementation environment.
- Discovery is a central task in SOA. SOA Discoverability is centered on Service Discoverability. Service Discoverability is meant to refer to the technology architecture's ability to provide a mechanism of discovery, for example a ***service directory, service registry or a service search engine***.
- Services be designed as resources that are highly discoverable in some fashion. Each service should be equipped with the metadata that is required to properly communicate its capabilities and meaning.

# SOA Principles – Composability

- Allow us to chain services together to provide new services
- Composition has the advantage that one can put together composite applications at a speed greater than writing one from scratch
- Building new services and application becomes quicker and cheaper

# SOA Properties – Self- Properties

- Most service architectures aim for „self- “ properties to reduce management load by design:
  - Self-Configuration
  - Self-Organization
  - Self-Healing
  - Self-Optimization
  - Self-Protection

# Self-Configuration

- Service architectures comprise of a huge amount of different components (services and hardware). Configuration is a challenging task in such environments.
- The idea of **self-configuration** is the adoption of the self-organization and fully distributed cooperation capabilities known from groups with cooperative social behavior which collaborate to solve a problem. Every member of the group can decide which part of the problem it can solve and which “QoS” it can provide.

# Self-Organization

- A system is **self-organizing** if it automatically, dynamically and autonomously adapts itself to achieve global goals more efficiently under changing conditions.

# Self-Healing

- The task of **self-healing** is to assure that a system meets some defined conditions as far as possible, i.e. to guarantee that all services running in the framework stay available, even in the case of partial outages in the system.

# Self-Optimization

- The **self-configuration** is responsible to find a good distribution of the services in terms of the given resources of the service description. The target of the **self-optimization** is to distribute the services of the application in a way that the considered resources are utilized evenly.
- A typical approach is to find an adequate configuration at the beginning and to optimize the application during runtime.



# Self-Protection

- **Self-protection** techniques cope with intentionally or unintentionally malicious peers or services in a framework. They behave as the “immune system” of a service framework as they are permissive to good-natured services and messages but can detect appearing malicious events.

# SOA Benefits

## **Business Benefits**

- Focus on Business Domain solutions
- Leverage Existing Infrastructure
- Agility

## **Technical Benefits**

- Loose Coupling
- Autonomous Service
- Location Transparency
- Late Binding

# Security Issues in the Cloud

- In theory, minimizing any of the issues would help:
  - Loss of Control
    - Take back control
      - Data and apps may still need to be on the cloud
      - But can they be managed in some way by the consumer?
  - Lack of trust
    - Increase trust (mechanisms)
      - Technology
      - Policy, regulation
      - Contracts (incentives): topic of a future talk
  - Multi-tenancy
    - Private cloud
      - Takes away the reasons to use a cloud in the first place
    - Strong separation