Studiengang Informatik

# CLOUD COMPUTING AUS DER SICHT DES ANWENDUNGSARCHITEKTEN

Open Cloud Day 2014

http://www.ch-open.ch/opencloudday

Prof. Dr. Olaf Zimmermann
Institute for Software, HSR FHO
Bern, 10. Juni 2014

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

# Abstract

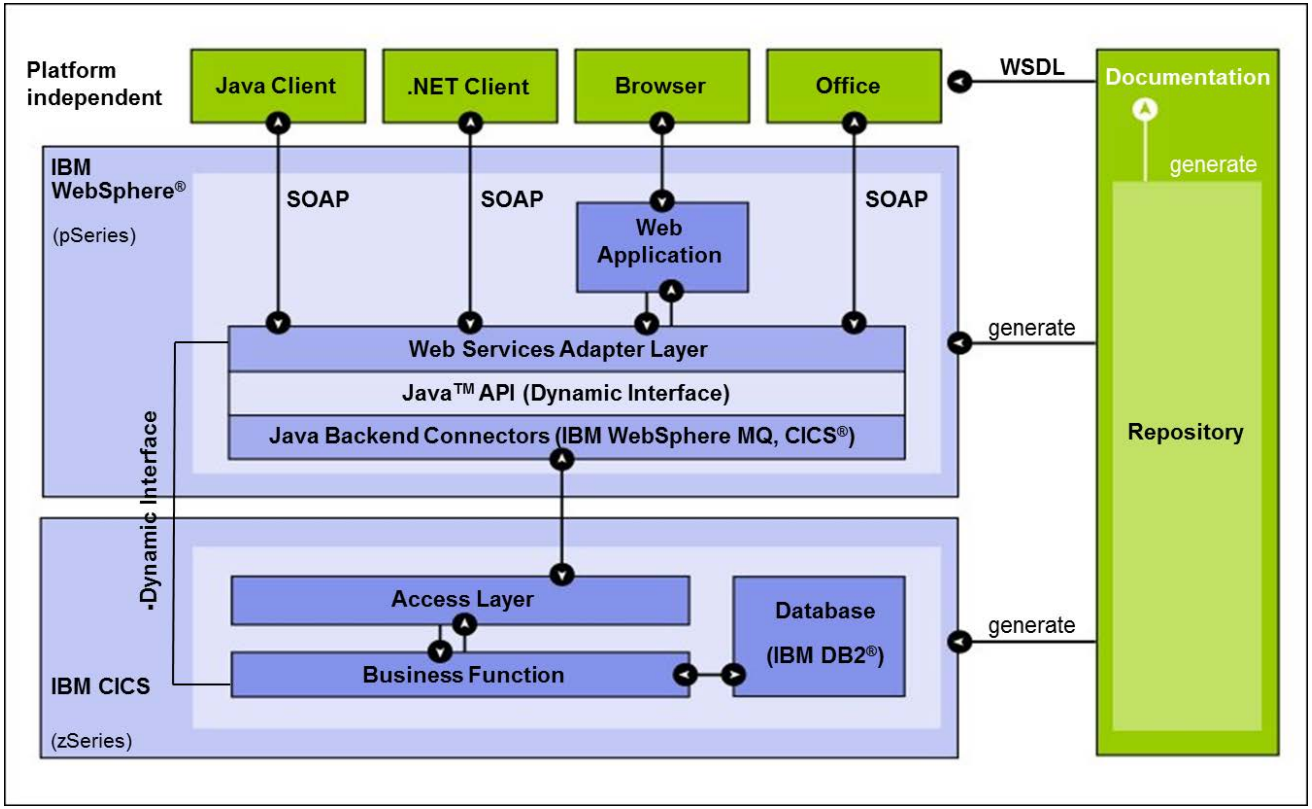

- **Architekten und Entwickler, die Cloud-Services nutzen wollen, sind mit einer Vielzahl neuer Designoptionen konfrontiert, z.B. nichtrelationale Speichertechniken (NoSQL), Message-Oriented Middleware mit At-Least-Once Delivery und Virtualisierung. Nicht alle klassischen Entwurfsmuster eignen sich für Cloud-Anwendungen; mit den Cloud-Ressourcen muss sparsam und fehlertolerant umgegangen werden. Cloud-Anbieter unterscheiden sich stark hinsichtlich ihrer Preismodelle, der zugesicherten Dienstgüten (Service Level Agreements) sowie der zur Verfügung stehenden Programmier- und Managementschnittstellen.**
- **Dieser Vortrag etabliert ausgewählte Cloud Computing Konzepte anhand von Architekturmustern, stellt wichtige Designoptionen und Praktiken bei ausgewählten Cloud-Anbietern vor und zeigt, wie Anwendungsarchitekturen cloudfähig gemacht werden können. Schwerpunkte werden dabei die Themen herstellerunabhängiger Entwurf und Architectural Refactoring for Cloud darstellen.**

# Agenda

- **Kontext und Motivation**
  - OSSM-Definition
- **Cloud Computing Patterns**
  - Cloud Offerings
  - Cloud Application Architecture
  - Cloud Management
- **IDEAL Cloud Application Architectures**
  - Amazon Best Practices, Twelve Factor App (Heroku)
- **Cloud Architecture Design and Cloud Migration – HSR Projects**
  - Cloud Deployment and Architectural Refactoring (CDAR) Lab
  - Architectural Refactoring for Cloud

# Gedankenexperiment: Ist dieses verteilte Informationssystem cloudfähig?

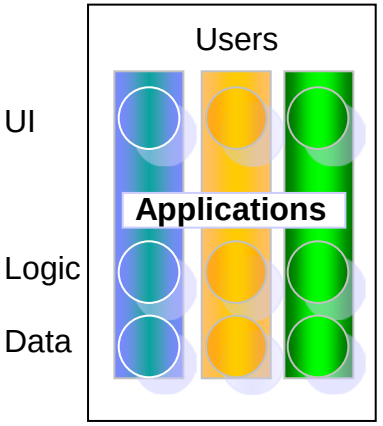
- **Core Banking Anwendung, Shared Service/Service Provider Modell**
  - Layers Pattern, Datenhaltung im Backend, Web Frontend, Web Services




**Referenz:** IBM, ACM OOPSLA 2004

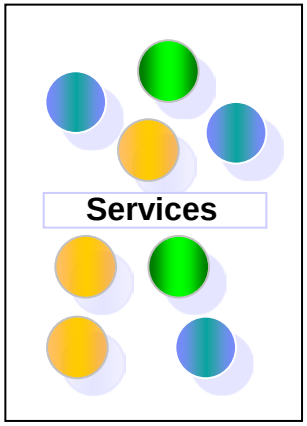
# From Traditional Layer-Tier Architectures to Cloud Services

**Traditional**




**Discrete Applications (Two or Three Tiers)**

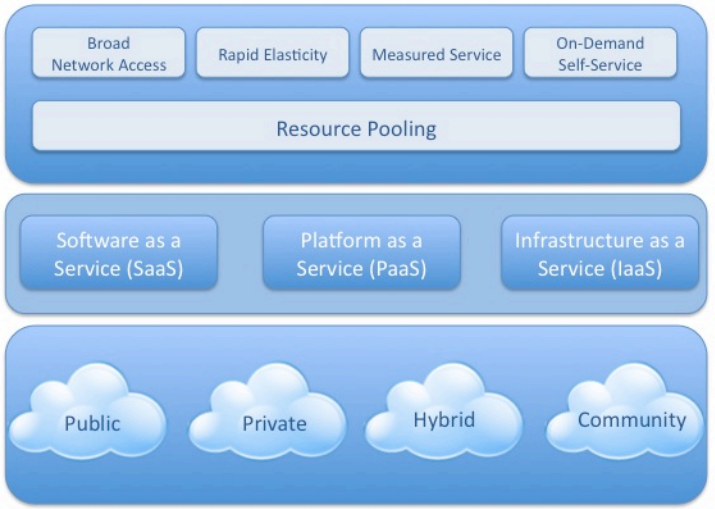
**SOA**




**Basket of Services**

?

Visual Model Of NIST Working Definition Of Cloud Computing
*http://www.csrc.nist.gov/groups/SNS/cloud-computing/index.html*



# Simple, User-Centered Definition of Cloud Computing

**Cloud computing provides a set of computing resources with the following testable characteristics:**

1. *On-demand*: the server is already setup and ready to be deployed (so the user can sign-up for the service without waiting)
2. *Self-service*: customer chooses what they want, when they want it (the user can use the service anytime, without waiting)
3. *Scalable*: customer can choose how much they want and ramp up if necessary (the user can scale-up the service when needed, without waiting for the provider to add more capacity)
4. *Measurable*: there's metering/reporting so you know you are getting what you pay for (the user can access measurable data to determine the status of the service)

**In summary, cloud computing is OSSM (pronounced 'awesome').**

**Reference:** B. Kepes, CloudU (online training, sponsored by RackSpace), Dave Nielsen, Cloud Camps, http://www.daveslist.com

# Risks and Inhibitors

- **Control over hosting environment**
  - Number of virtual nodes, upgrade policy
- **Data privacy and regulation**
  - E.g. see http://www.datenschutz-forum.ch/ (Switzerland)
- **Legal responsibilities, compliance, Service Level Agreements (SLAs)**
  - Who is responsible for outages, who has to proof what happened?
  - How is troubleshooting done?
  - Do desaster recovery plans exist?
- **Single point of failure, external dependencies**
  - Which IaaS Provider is used by PaaS- or SaaS-provider (how about indirect dependencies)?
  - How about future dependencies?
  - What happens in case of mergers and acquisitions (on provider side)?

# PaaS Platform Selection/Evaluation Criteria (1/2)

- **Support for defining cloud characteristics (“OSSM”):**
  - On demand, Self service, Scalable, Measured
- **Billing model and Service Level Agreements (SLAs)**
  - Accountability of provider, penalties/refunds, customer obligations
- **Physical location (of data)**
- **Country/place of jurisdiction (CH/EU/other)**
- **Service scope**
  - Platform middleware versions?
  - Limitations:
    - Can main programs (batch jobs) be run?
    - Can JEE EARs be deployed?

# PaaS Platform Selection/Evaluation Criteria (2/2)

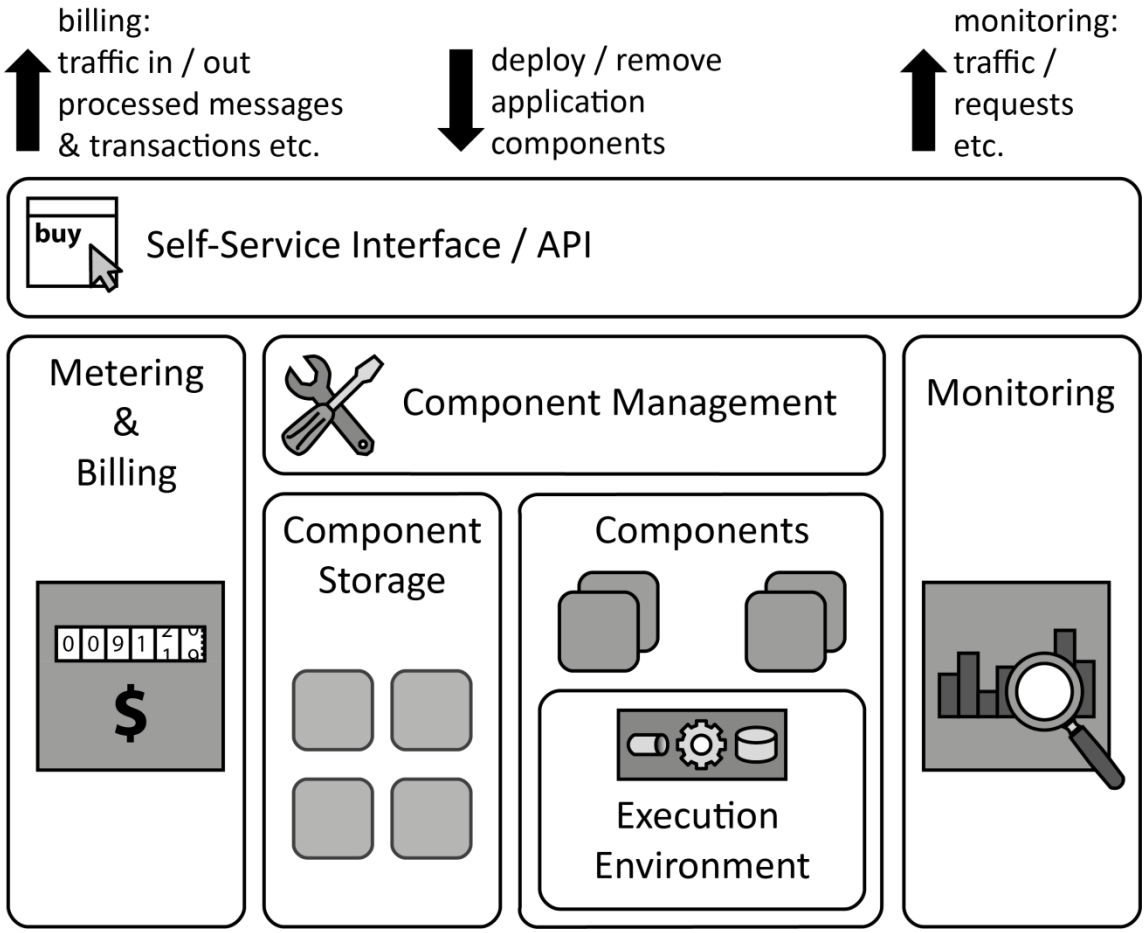
- **Deployment process and tools; standardization**
  - Web console
  - Management APIs
  - Local SDK (command line tools, Eclipse plugins)
  - Topology and Orchestration Specification for Cloud Applications (TOSCA)
- **User/programmer documentation incl. getting started information**
- **Cloud services lifecycle, e.g. hibernation due to inactivity/restart time?**
- **Operational model (runtime topologies)**
  - Inbound traffic, outbound traffic, cloud-internal communication
- **Domain and port management capabilities for user**
  - E.g. own URIs/domain names possible (DNS management)?
  - Can virtual hosts (custom DNS entries) be defined?
- **API security and VPN support**
  - Credentials, storage locations

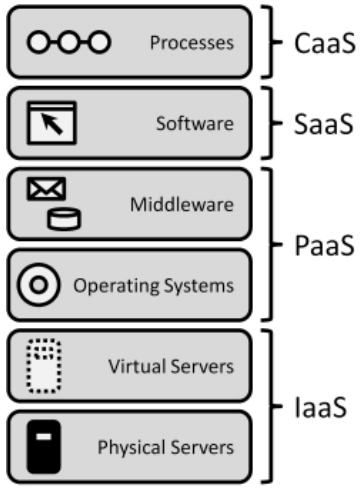
# SLA Questions to Ask (Source: Dimension Data White Paper)

- **Areas covered: e.g. availability, support**
- **Questions to ask: scope (SLO), provider-side protection means, measurements, penalties, actions required**
- **Conclusion – do SLAs really matter?**

https://www.dimensiondata.com/Global/Downloadable%20Documents/Comparing%20Public%20Cloud%20Service%20Level%20Agreements%20White%20Paper.pdf

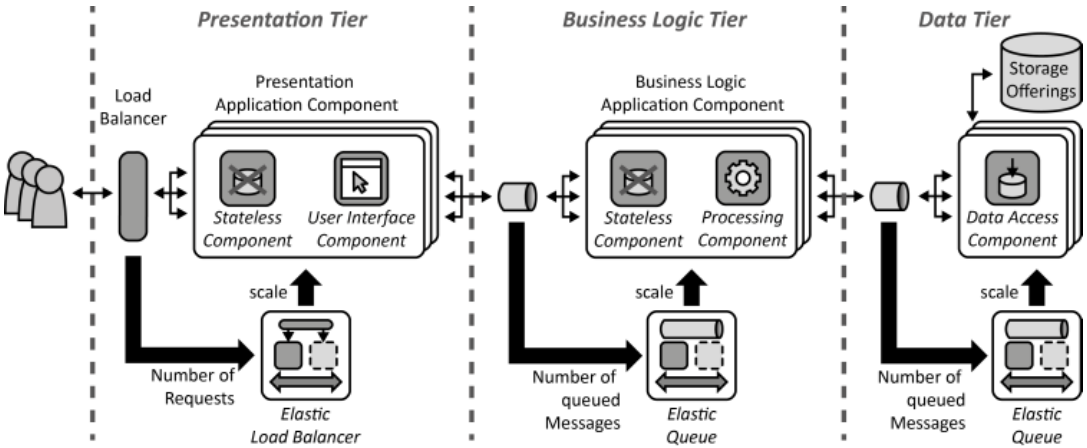
# Cloud Computing Patterns (CCP)


**Reference:** Cloud Computing Patterns, Springer 2014, http://cloudcomputingpatterns.org/

# Cloud Application Components (Source: CCP)




- http://www.cloudcomputingpatterns.org/Category:Cloud_Application_Components

**Cloud Application Architectures**

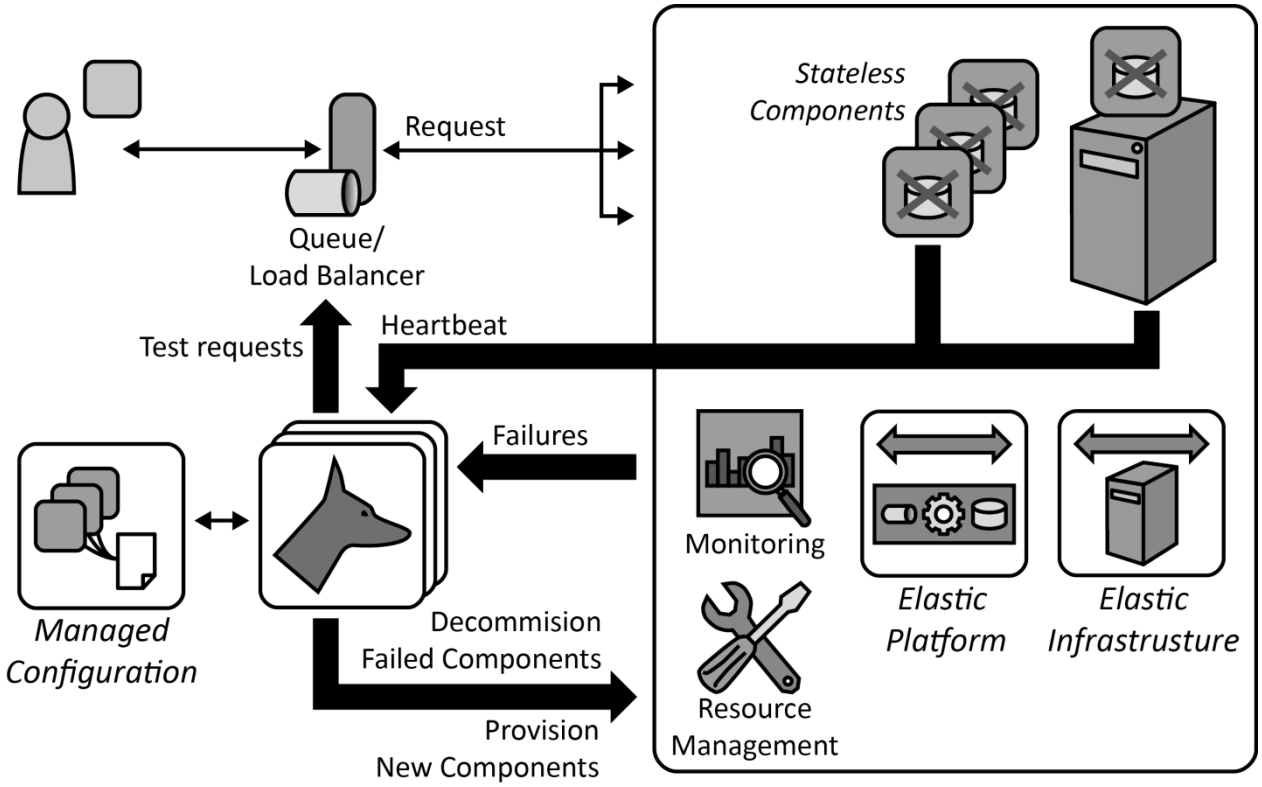
Fundamental Cloud Architectures
- Loose Coupling
- Distributed Application

Cloud Application Components
- Stateful Component
- Stateless Component
- User Interface Component
- Processing Component
- Batch Processing Component
- Data Access Component
- Data Abstractor
- Idempotent Processor
- Transaction-based Processor
- Timeout-based Message Processor
- Multi-Component Image

Multi-Tenancy
- Shared Component
- Tenant-isolated Component
- Dedicated Component

Cloud Integration
- Restricted Data Access Component
- Message Mover
- Application Component Proxy
- Compliant Data Replication
- Integration Provider

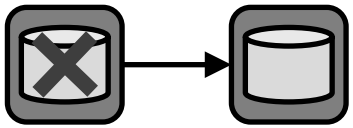
# Watchdog Pattern



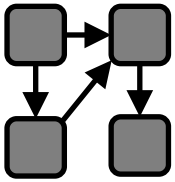

- Application components are *stateless*
- Component health is monitored
  - **Periodic heartbeats**: components notify that they are functioning
  - **Test requests**: result of test data is compared to expected results
  - **Environment**: provider-supplied reachability monitoring

# IDEAL Cloud Application Properties (Fehling et al.)

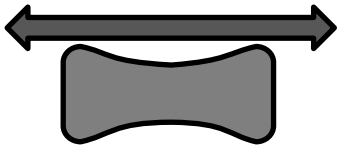
**Reference:** Cloud Computing Patterns, Springer 2014, http://cloudcomputingpatterns.org/



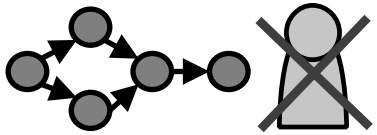
**Isolated State:** most of the application is *stateless* with respect to:
*Session State*: state of the communication with the application
*Application State*: data handled by the application



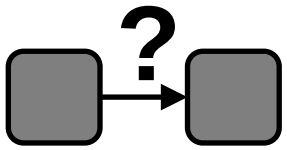
**Distribution:** applications are decomposed to…
… use multiple cloud resources
… support the fact that clouds are large globally distributed systems



**Elasticity:** applications can be scaled out dynamically
*Scale out*: performance increase through addition of resources
*Scale up*: performance increase by increasing resource capabilities



**Automated Management:** runtime tasks have to be handled quickly
Example: exploitation of pay-per-use by changing resource numbers
Example: resiliency by reacting to resource failures




**Loose Coupling:** influence of application components is limited
Example: failures should not impact other components
Example: addition / removal of components is simplified

# Best Practices für Cloud-Native Applications

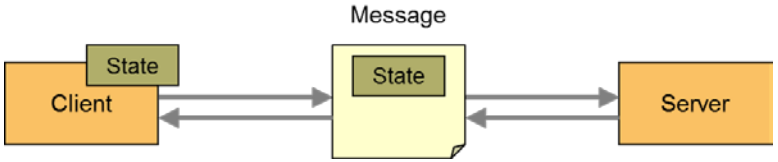
- **J. Varia from Amazon (Reference: http://aws.amazon.com/whitepapers/)**
  - **“Design for failure and nothing will fail**
  - **Decouple your components**
  - **Implement elasticity**
  - **Think parallel**
  - **Keep dynamic data closer to the compute and static data closer to the end-user”**

# The Twelve-Factor App (Source: Heroku Co-Founder)

https://blog.heroku.com/archives/2013/8/15/twelve-factor-apps

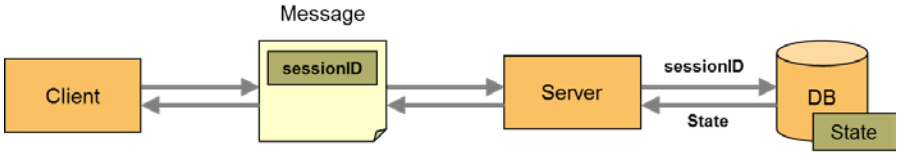

- **Not cloud-specific**
- **Mix of agile and DevOps practices (more on DevOps next week)**
- **In line with IDEAL, Amazon, ARC/CDAR**

# Session State Management Design – Options




- **Client Session State**
  - Scales well, but has security and possibly performance problems
  - This does not change when moving to a cloud platform.


- **Server Session State**
  - Uses main memory or proprietary data stores in an application server (e.g. HTTP session in JEE servlet container)
  - Persistent HTTP sessions no longer recommended when deploying to a cloud due to scalability and reliability concerns.




- **Database Session State**
  - Is well supported in many clouds, e.g. via highly scalable key-value storage (a type of NoSQL database)

# Cloud Affinity of PoEAA Patterns (1/3)

| PoEAA Pattern | Suitability for Cloud | Comment |
|---|---|---|
| Client Session State | Yes and no | As good or bas as in traditional deployment (security?) |
| Server Session State | No (I in IDEAL violated) | Also hinders scale out |
| Database Session State | Yes | Can use DB (e.g. NoSQL) |
| Model-View-Controller | Yes (with persistent model) | Web frontends are cloud-affine |
| Front Controller | Yes (Web frontends) | See above |
| Page Controller | Yes (Web frontends) | See above |
| Application Controller | Yes (Web frontends) | See above |
| other Presentation Layer Patterns | Yes (Web frontends) | See above |

**Patterns of Enterprise Application Architecture Patterns (PoEAA):**
http://martinfowler.com/eaaCatalog/

# Cloud Affinity of PoEAA Patterns (2/3)

| PoEAA Pattern | Suitability for Cloud | Comment |
|---|---|---|
| Transaction Script | Yes | Procedures should be self contained (stateless interactions) |
| Domain Model | Depends on complexity of domain model | Object tree in main memory might limit scale out (and database partitioning) |
| Table Module | No or implementation dependent | Big data sets problematic unless partitioned (e.g. map-reduce) |
| Service Layer | Yes | SOA and REST design principles should be adhered to, e.g. no object references in domain model, but only instances of Data Transfer Object in interface (larger discussion required) |
| Remote Facade | Yes | Can be introduced for cloud enablement of existing solutions; can wrap calls to PaaS provider to support maintainability and portability |

**Patterns of Enterprise Application Architecture Patterns (PoEAA):**
http://martinfowler.com/eaaCatalog/

# Cloud Affinity of PoEAA Patterns (3/3)

| PoEAA Pattern | Suitability for Cloud | Comment |
|---|---|---|
| Active Record | Limited | Good when RDB exists in cloud or when records have simple structures; complex structures can be difficult to handle for NoSQL storage (mapping need) |
| Row Data Gateway | Yes | Fits scale out |
| Table Data Gateway | No or implementation dependent | Big data sets problematic unless partitioned (e.g. map-reduce) |
| System Transaction | Depends on cloud storage capabilities (NoSQL?) | Larger discussion required (CAP BASE vs. ACID etc.) |
| Business Transaction | Yes | If cloud design best practices are adhered to (statelessness etc.) |

**Patterns of Enterprise Application Architecture Patterns (PoEAA):**
http://martinfowler.com/eaaCatalog/

# Architectural Principles for Cloud-Native Applications

- **Design application startup and restart procedures as lean as possible**
  - How long does it take your application server to display an “open for e-business” message after a restart (process and/or hardware)?
- **Let all components implement the Service Layer pattern**
  - Define with Remote Facades and expose them with JAX-WS or JAX-RS
  - Use messaging for cloud-internal communication and integration
- **Define all Data Transfer Objects (DTOs) to be serializable**
  - See experiment with DDD Sample in PaaS Provider 1 (Spring MVC)
- **Use Internet security technologies to satisfy application security needs**
  - E.g. often no connectivity to company-internal LDAP or Active Directory
- **Model all communication dependencies explicitly and consult IT infrastructure architects both on provider and on consumer side**
  - E.g. one PaaS Provider requires inbound port 5000 connectivity to support remote terminals (required for platform/instance management)

# Good Cloud Design Practices

- **Avoid calls to proprietary platform libraries (e.g., via JNI)**
- **Limit usage of expensive operations, e.g. SecureRandom in Java SE**
- **Do not define resource identifiers such as IP addresses statically**
- **Prefer HTTP over raw socket communication even for cloud-internal integration (or use messaging capabilities offered by cloud provider)**
- **Do not expect cloud messaging to have the same semantics and QoS as traditional messaging systems (at-least-once vs. exactly-once delivery)**
- **Do not expect NoSQL storage to provide the same level of programming and database management convenience as mature SQL database systems**
- **Do not expect cloud provider to handle backup and recovery of application data for you**
- **Be prepared to log resource consumption on same level of detail as provider (in case bill from provider contains suspicious items)**

# QuoVadis, xaaS?

- **Virtualization is here to stay**
- **Shift from Capex to Opex is inevitable (only pay for what you use)**
- **SLAs and pricing schemes not yet adequate (for many applications)**

- **IaaS is established and will continue to grow**
  - Cloud sceptic: http://cramer.io/2012/06/02/the-cloud-is-not-for-you/
- **PaaS is challenged – need to provide more than IaaS VM plus pre-installed open source middleware**
  - PaaS market will consolidate (just like JEE application server market)
- **SaaS is a KMU opportunity (provider side, consumer side)**
- **Private cloud and cloud management software still emerging**
  - Critical voices: e.g. this blog post (on private clouds) and this one (on OpenStack)

# Schlussgedanken (1/2)

- **Cloud Computing wird zukünftig *ein* wichtiges Hostingmodell sein**
  - Cloud ist OSSM (pronounce: ”awesome”)
  - Utility computing made real – nach vielen vergeblichen Anläufen
- **Definierende Konzepte:**
  - Service-Modelle, Deployment-Modelle
  - SLAs und Billing (Opex statt Capex)
- **Plattformneutrale, herstellerunabhängige Cloud Computing Patterns (CCP) sind jetzt in der Literatur beschrieben:**
  - Cloud Offerings, Application Architectures, Watchdog
  - Weitere Patterns: At-Least-Once Delivery, Map-Reduce, Key-Value Storage
- **Cloud Design ist Software-Architektur-Design**
  - Viel Management-Bedarf, Security-Fragestellungen
- **HSR-Projekte: CDAR Lab, ARC, IFS Software Health Check (for Cloud)**

# Schlussgedanken (2/2)

- **Der cloud-interessierte Anwendungsarchitekt…**
  - Kennt CPU- und Speicherverbrauch seiner Anwendung
  - Kennt die SLAs seiner Cloud-Provider Shortlist
  - Trifft bewusste Architekturentscheidungen (Cloud-Patternwahl usw.)
- **Der cloud-freundliche Infrastrukturarchitekt (bzw. Betriebsleiter)…**
  - Kann Anwendungen nicht nur auf eigene, sondern auch auf externe Clouds deployen
  - Betreibt Private Cloud für sein Unternehmen/seine Organisationseinheit
  - Beherrscht das automatisierte Cloud Provisioning und überwacht Deployments mit Hilfe von Watchdogs etc.
- **Der cloud-informierte Unternehmensarchitekt…**
  - Hat eine Cloud-Strategie
  - Beachtet Data Privacy-Randbedingungen
  - Bietet Cloud Coachings und Readyness Assessments an

# More Information

- **CDAR and ARC projects at HSR**
  - http://www.ifs.hsr.ch/Olaf-Zimmermann.11623.0.html?&L=4 and ozimmerm@hsr.ch
- **Online-Schulung**
  - E.g. Rackspace Cloud University (CloudU), http://www.rackspace.com/knowledge_center/cloudu/
- **Cloud Computing Patterns (Springer 2014)**
  - http://cloudcomputingpatterns.org/?page_id=305
- **Analysten-Reports und Knowledge Hubs**
  - z.B. InfoWorld, DZone
- **Blogs**
  - http://searchcloudcomputing.techtarget.com/feature/Top-five-must-read-cloud-computing-blogs