

Enterprise Computing Lab



CLOUD COMPUTING – AUS DER SICHT DES ANWENDUNGSARCHITEKTEN



IFS

INSTITUTE FOR
SOFTWARE

Prof. Dr. Olaf Zimmermann

Distinguished (Chief/Lead) IT Architect, The Open Group

ozimmerm@hsr.ch

München, 4. Februar 2014



HSR

HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

■ Kontext:

- Architekten und Entwickler, die Cloud Computing 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 Entwurfsmuster eignen sich für Cloud-Anwendungen; mit den Cloud-Ressourcen muss sparsam und fehlertolerant umgegangen werden.

■ Dieser Vortrag:

- etabliert Cloud Computing Konzepte anhand von Architekturmustern,
- stellt wichtige Designoptionen bei ausgewählten Cloud-Anbietern vor und
- zeigt, wie Anwendungsarchitekturen cloudfähig gemacht werden können.

■ R+D und Professional Services-Erfahrung seit 1994

- em. IBM Executive IT Architect (senior certified by The Open Group)
 - Systems & Network Management, J2EE, Enterprise Application Integration/SOA
- em. ABB Senior Principal Scientist
 - Enterprise Architecture Management/Legacy System Modernization/Remoting

■ Diverse Industrieprojekte und Coachings

- R+D und IT-Consulting für Middleware, Informationssysteme, Tools
- Tutorials: UNIX/RDBMS, OOP/C++/J2EE, MDSE/MDA, SOA/XML

■ Schwerpunkt @ HSR: Entwurf verteilter Systeme

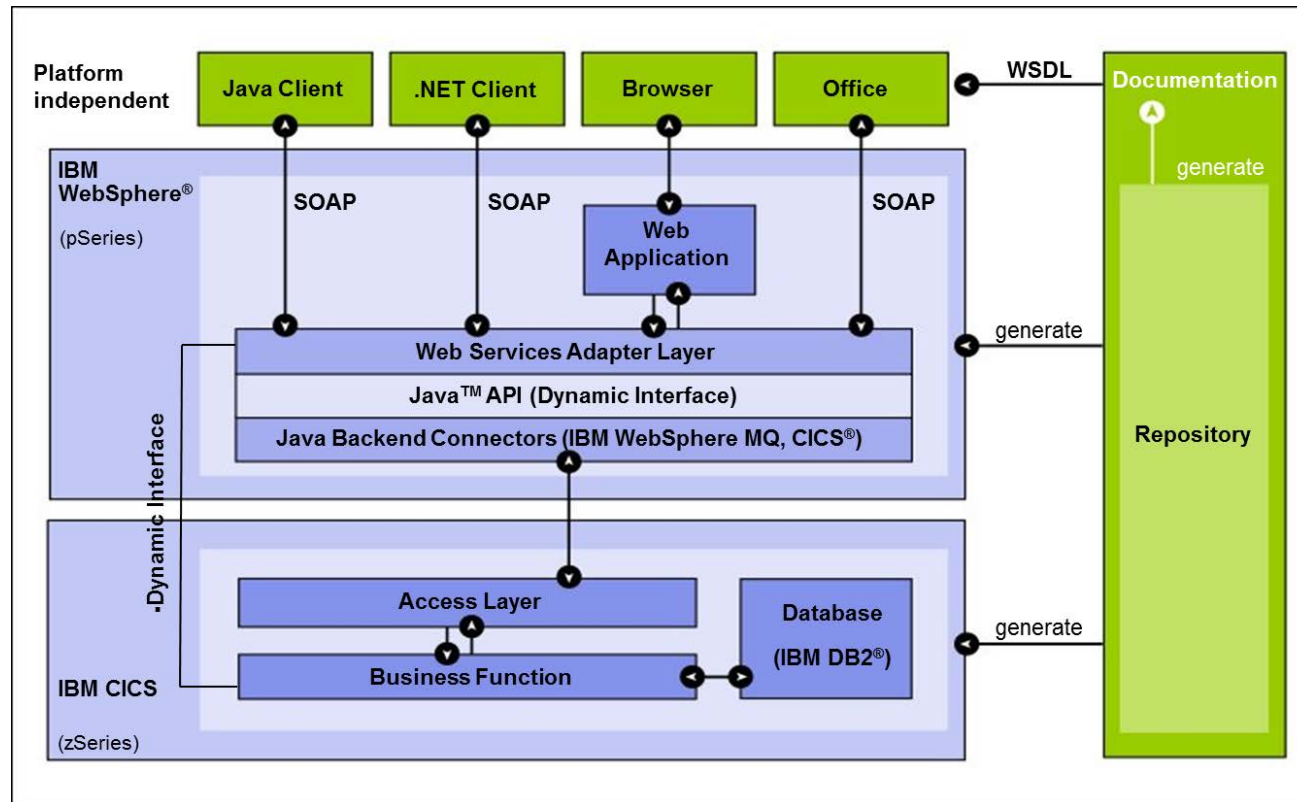
- Cloud, SOA, Web Application Development (Runtime)
- Architekturentscheidungen, Architectural Refactoring (Build Time)

Aktivitäten im Bereich Cloud Computing

- **Pre-Cloud Age: Beratung zu JEE, SOA und e-business Infrastrukturen**
 - Finanzdienstleister und Zentralbank (D); Corporate IT Telco-Provider (CH)
- **Architekturentscheidungen und -prinzipien in Cloud-Referenzarchitektur**
 - Diese Architektur wurde später zur IBM-Einreichung bei The Open Group
 - Architectural Decision (AD) Guidance Model präsentiert auf der SEI-Konferenz SATURN 2011
- **Herstellerevaluation und PoC im Rahmen eines firmeninternen EAM-Projektes in Prozessautomatisierungs- und Energietechnikkonzern**
 - Software-as-a-Service, Microsoft-Technologien
- **HSR-Projekte *Architectural Refactoring for Cloud (ARC)* und *Cloud Deployment and Architectural Refactoring Lab (CDAR)***
- **Stets ein Schwerpunkt: *Design for Operations* (vgl. DevOps-Bewegung)**

Thought Experiment: Is this Information System Cloud-Affine?

- Core banking system, shared service/service provider model
 - Layers Pattern, data and logic in backend, Web frontend, Web services



Reference: IBM, ACM OOPSLA 2004

Agenda

■ Cloud computing fundamentals

- Definitions
- Patterns
- Usage scenarios
- Risks and inhibitors

■ Design options at selected PaaS providers

- Decisions required
- Decision criteria
- Good design practices

■ Architecture design for cloud

- IDEAL application properties
- Pitfalls to avoid
- Cloud readiness assessment and architectural refactoring

Motivation: Famous First Words on Cloud...

- *“The interesting thing about cloud computing is that we’ve redefined cloud computing to include everything that we already do. I can’t think of anything that isn’t cloud computing with all of these announcements. The computer industry is the only industry that is more fashion-driven than women’s fashion. Maybe I’m an idiot, but I have no idea what anyone is talking about. What is it? It’s complete gibberish. It’s insane. When is this idiocy going to stop?”*
 - [Larry Ellison](#), chairman, Oracle
- *“If you think you’ve seen this movie before, you are right. Cloud computing is based on the time-sharing model we leveraged years ago before we could afford our own computers. The idea is to share computing power among many companies and people, thereby reducing the cost of that computing power to those who leverage it. The value of time share and the core value of cloud computing are pretty much the same, only the resources these days are much better and more cost effective.”*
 - [David Linthicum](#), author, Cloud Computing and SOA Convergence in Your Enterprise: A Step-by-Step Guide

Reference: J. McKendrick, 10 Quotes on Cloud Computing That Really Say it All, Forbes 2013

History and Enablers

- **Utility computing (paper ~1960s)**
- **Host virtualization (on the mainframe)**
- **on demand computing (IBM strategy following e-business ~2002)**

- **Omnipresence of the Web, success of Internet technologies**
 - Always-on mentality, WLANs, mobile computing, Web services, SOA
- **Commodity hardware (blade PCs)**
- **Advanced in automation (of systems management)**

Pragmatic Definition (by CloudCamp/Dave Nielsen)



“Cloud is OSSM (pronounced ‘awesome’), meaning that Cloud Computing is a computing resource that is:

1. *On-demand*: the server is already setup and ready to be deployed
2. *Self-service*: customer chooses what they want, when they want it
3. *Scalable*: customer can choose how much they want and ramp up if necessary
4. *Measurable*: there’s metering/reporting so you know you are getting what you pay for”

So not (only): Online Storage, Services, Web Hosting, Virtualization

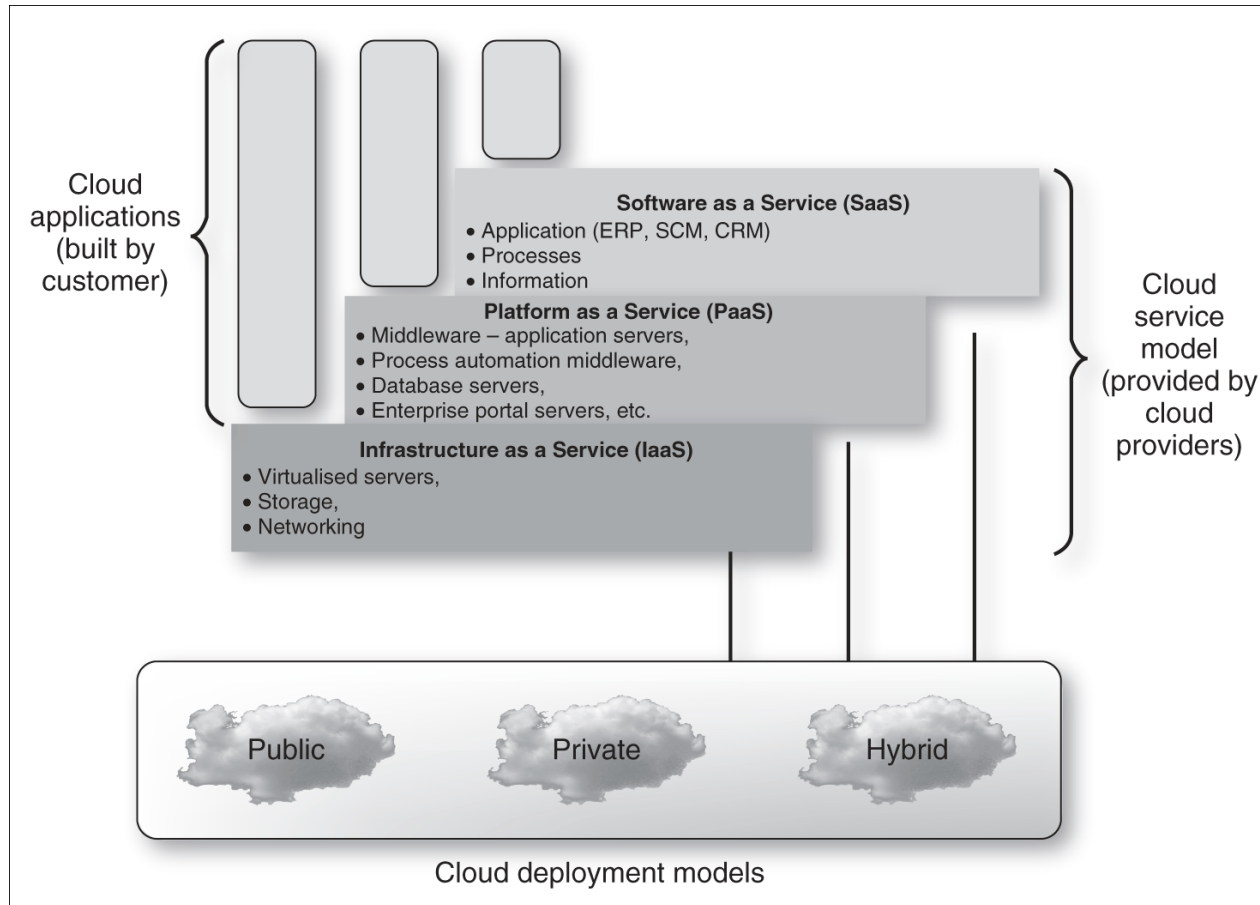
Reference: B. Kepes, CloudU (online training, provided by RackSpace)

Official Definition (NIST)

- **Cloud Computing is a model for enabling ubiquitous, 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.**
 - The essential characteristics are on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service.
 - The three service models are software, platform, and infrastructure (all as a service).
 - The four deployment models: private, community, public, and hybrid

Reference: The NIST Definition of Cloud Computing, <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>

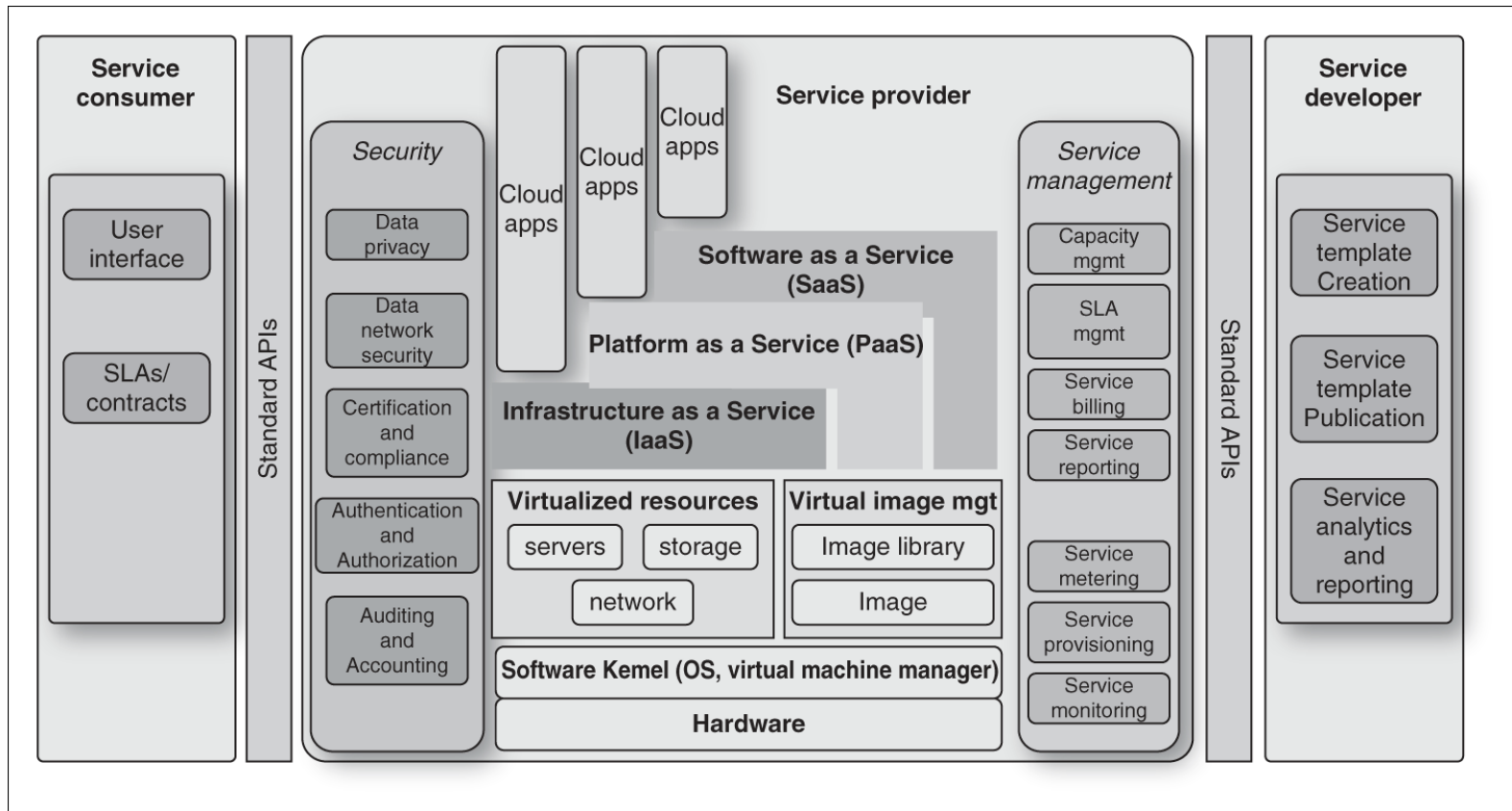
NIST Definition (Visualization)



Reference: M. Papazoglou, *Web Services 2e PowerPoints on the Web*, 2nd Edition, © Pearson Education Limited 2012


Functional Roles and Modules in a Cloud Architecture


- Roles: Service consumer, service provider, service developer





Reference: M. Papazoglou, *Web Services 2e PowerPoints on the Web*, 2nd Edition, © Pearson Education Limited 2012


Cloud Computing Provider Example: Amazon Web Services


 **Amazon EC2 »**
Web service that provides resizable compute capacity in the cloud.

 **Amazon S3 »**
Highly-scalable, reliable, and low-latency data storage.


 **Amazon RDS »**
Managed MySQL, Oracle and SQL Server databases.

 **Amazon CloudWatch »**
Monitoring for AWS cloud resources and applications.

 **AWS Data Pipeline »**
Orchestration for data-driven workflows.


 **Amazon DynamoDB »**
Fully managed NoSQL database service with seamless scalability.


 **Amazon EBS »**
Highly available, highly reliable, predictable storage volumes.


 **Amazon ELB »**
Web service that provides scalability and high availability.

 **Amazon ElastiCache »**
Managed scale-out caching.

 **Amazon SNS »**
Web service to set up, operate, and send notifications from the cloud.

 **Amazon Elastic Transcoder »**
Convert your media files easily, at low cost and at scale.

 **Amazon SQS »**
Scalable queue for storing messages as they travel between computers.

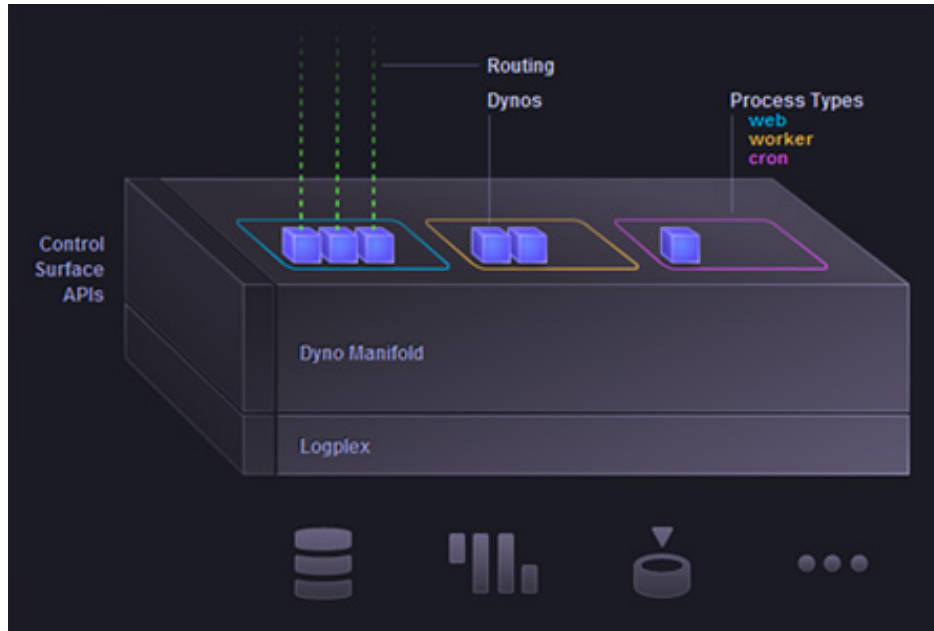
 **Amazon SWF »**
Workflow service for building scalable, resilient applications.

<http://www.deepfield.net/2012/04/how-big-is-amazons-cloud>
(2012)

AWS Now Five Times The Size Of Other Cloud Vendors Combined (Gartner 2013)

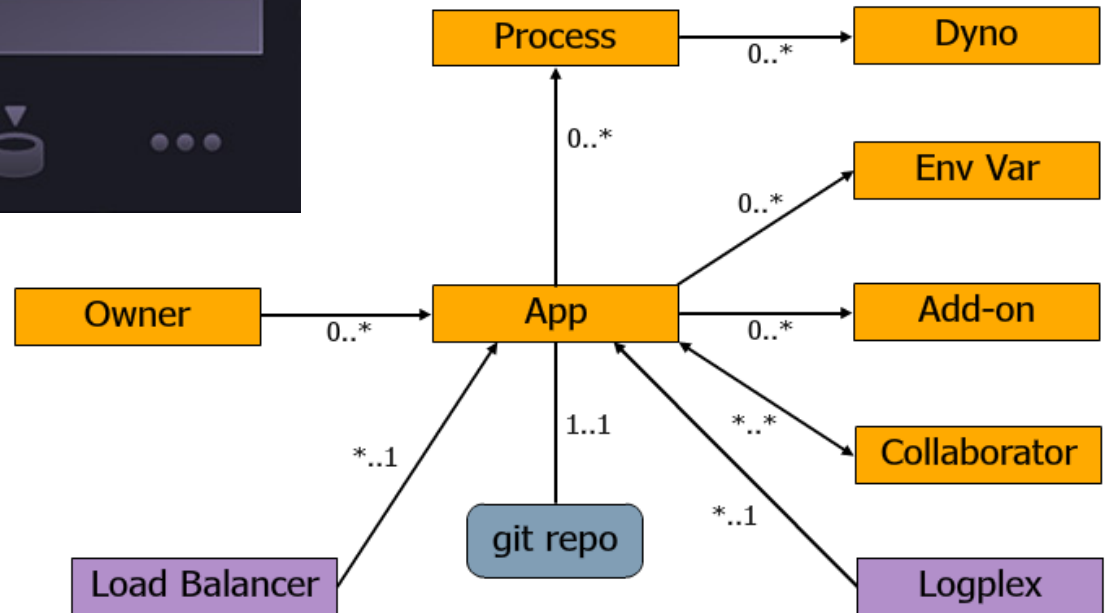
<http://readwrite.com/2013/08/21/gartner-aws-now-5-times-the-size-of-other-cloud-vendors-combined>

Platform-as-a-Service (PaaS) Provider Example: Heroku



- **Measured: Add On features and *Dyno* use**

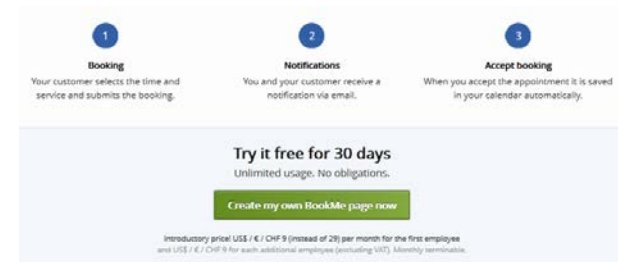
- Kind of virtual UNIX shell: <https://devcenter.heroku.com/articles/dynos>



Software-as-a-Service (SaaS) Example: BookMe (Doodle)

- **Multi-tenant Web application that provides a customer self service channel for booking service appointments**
 - Particularly interesting for small and medium businesses like hairdressers, therapists, driving schools (SaaS consumers)
 - Integrated with electronic calendars like Google Calendar
 - No need to install any software (for SaaS consumers and their clients)
 - Supplied/provided by Doodle AG (SaaS provider)
 - Domain expertise (business process: calendar scheduling)
 - User interface and application logic
 - High-scale IT infrastructure
 - Subscription-based monthly billing

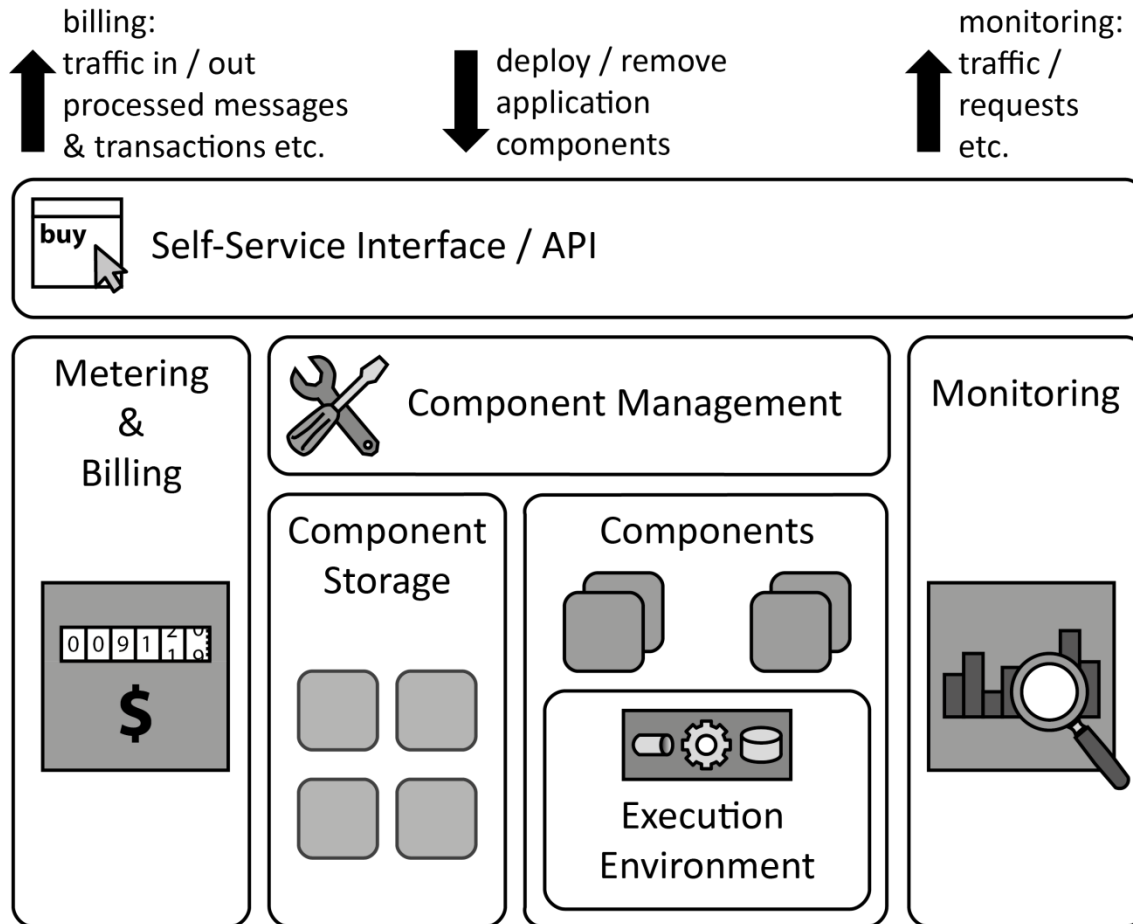
BookMe
powered by Doodle



■ <https://doodle.com/bookme/>

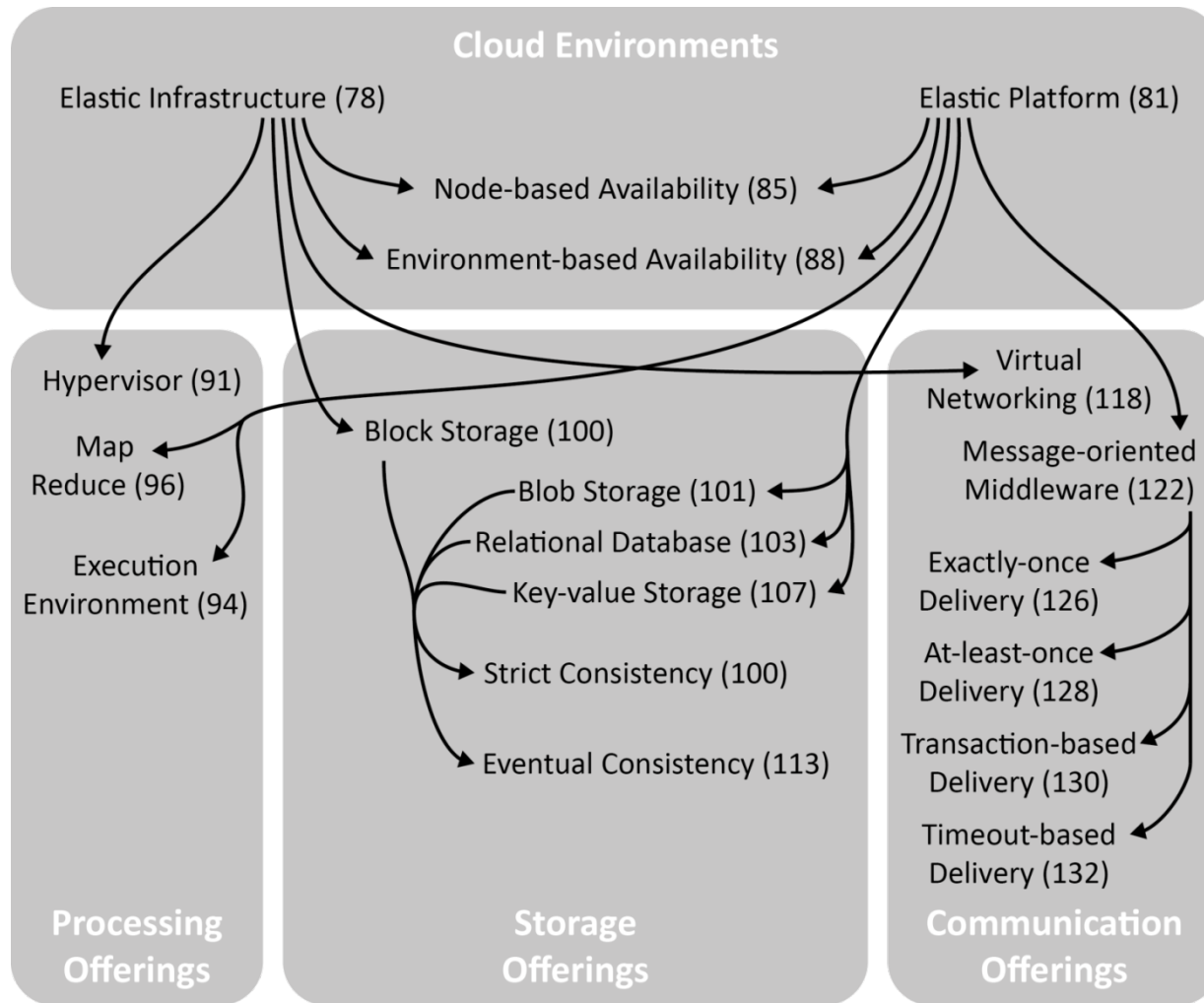
■ <http://en.blog.doodle.com/2013/11/18/doodles-technology-landscape-2/>

Cloud Computing Patterns (Springer-Verlag 2014)

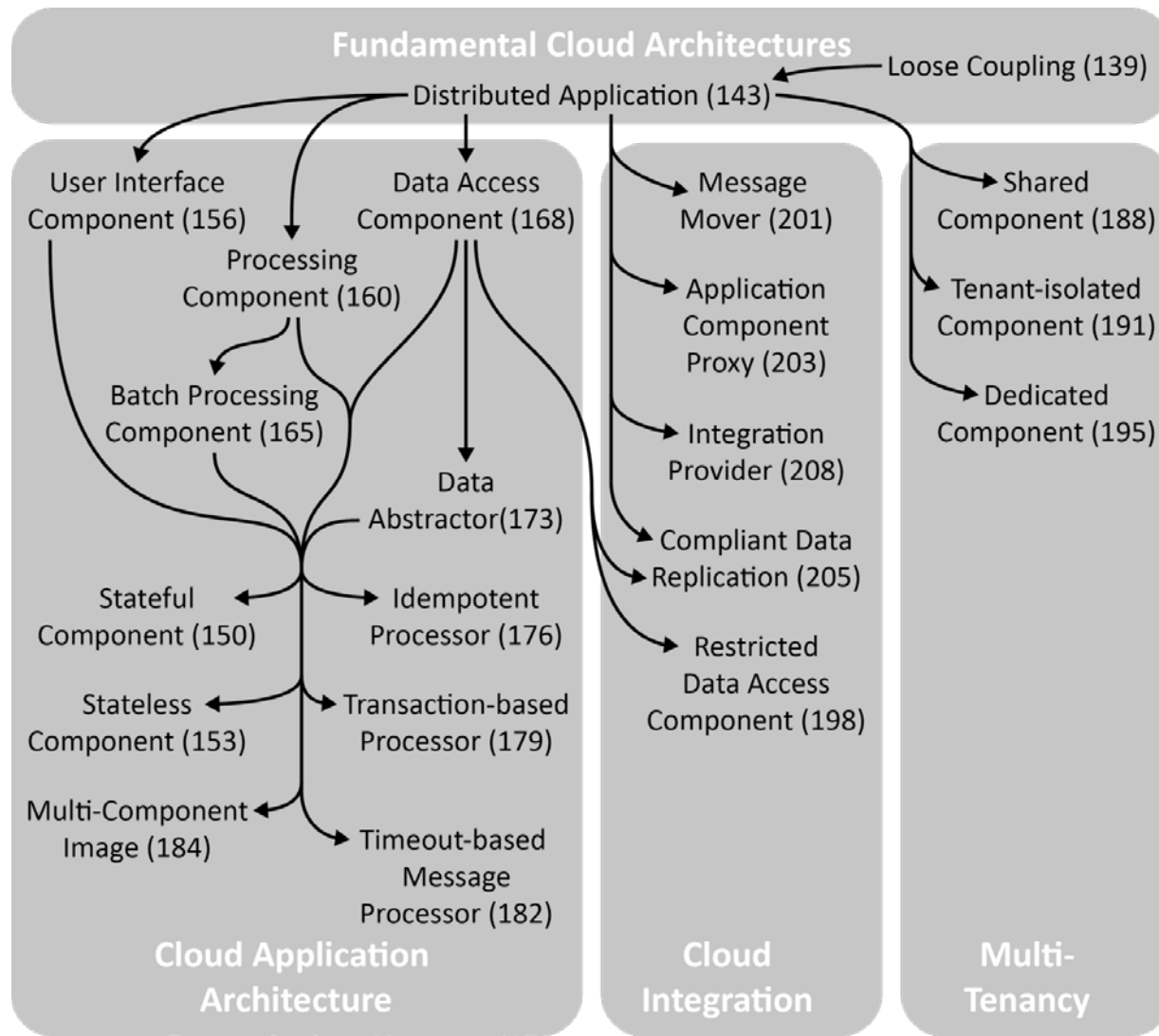


Reference: Cloud Computing Patterns, Springer 2014, <http://cloudcomputingpatterns.org/>

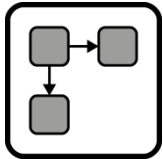
Pattern Map: Cloud Offerings



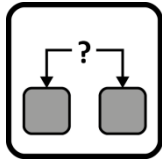
Pattern Map: Application Architecture Patterns



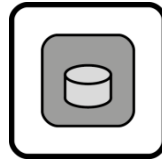
Cloud Application Architecture Patterns (Overview)



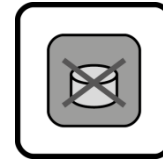
Distributed Application



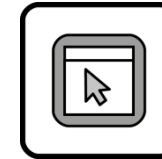
Loose Coupling



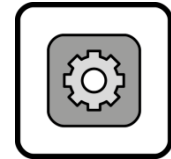
Stateful Component



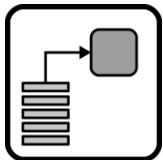
Stateless Component



User Interface Component



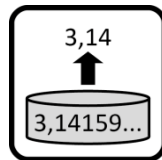
Processing Component



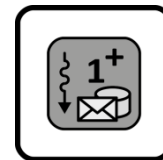
Batch Processing Component



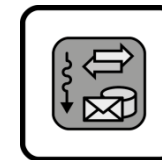
Data Access Component



Data Abstractor



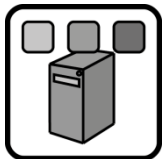
Idempotent Processing Component



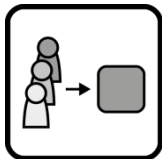
Transaction-based Processor



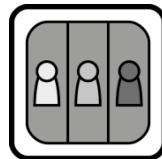
Timeout-based Message Processor



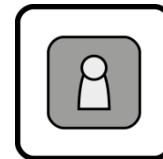
Multi-component Image



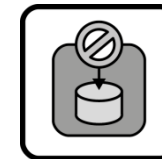
Shared Component



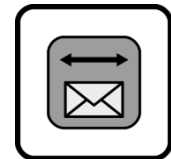
Tenant-isolated Component



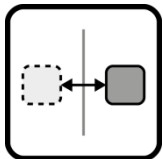
Dedicated Component



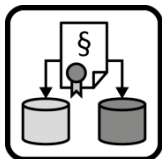
Restricted Data Access Component



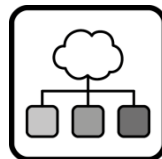
Message Mover



Application Component Proxy



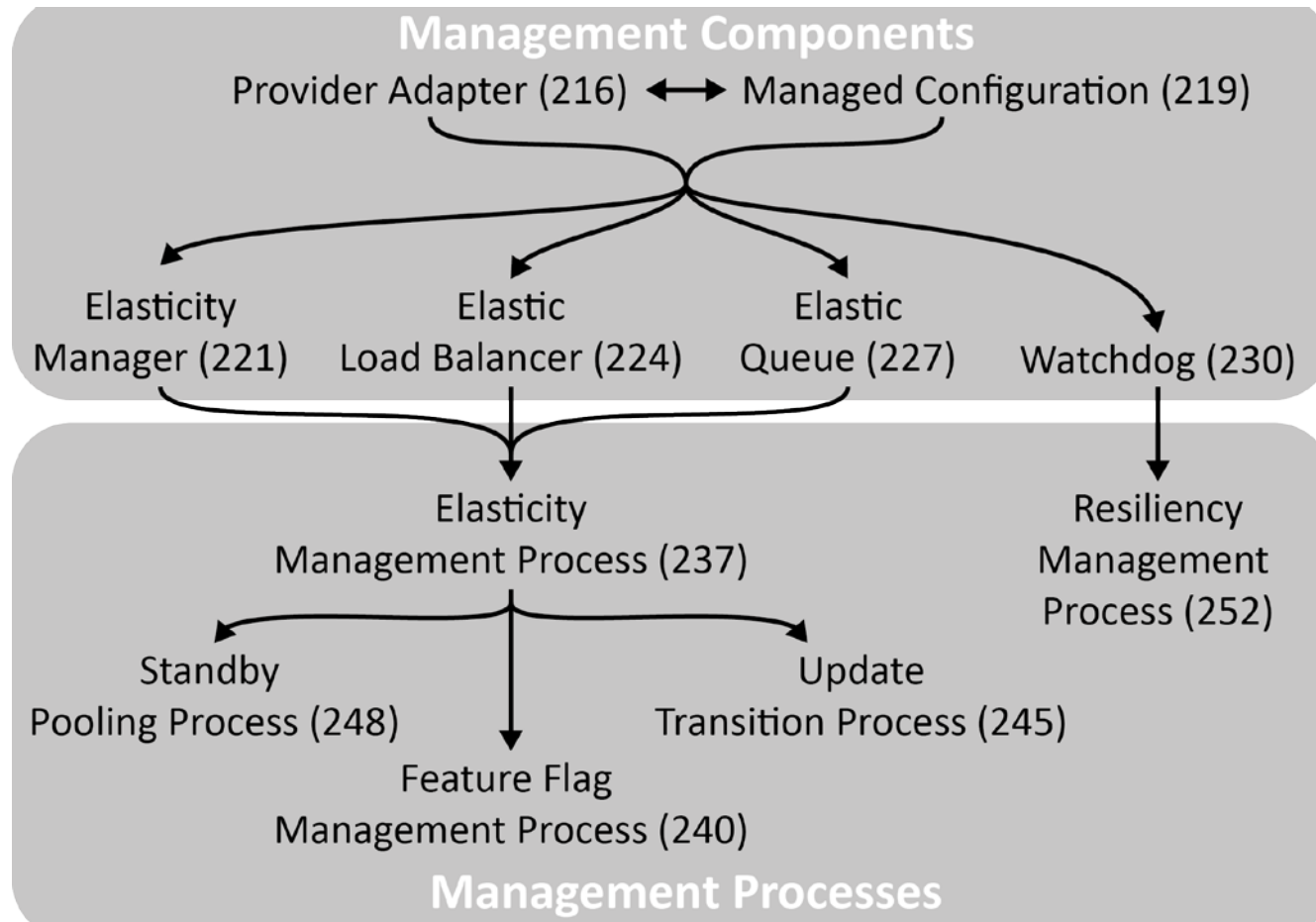
Compliant Data Replication



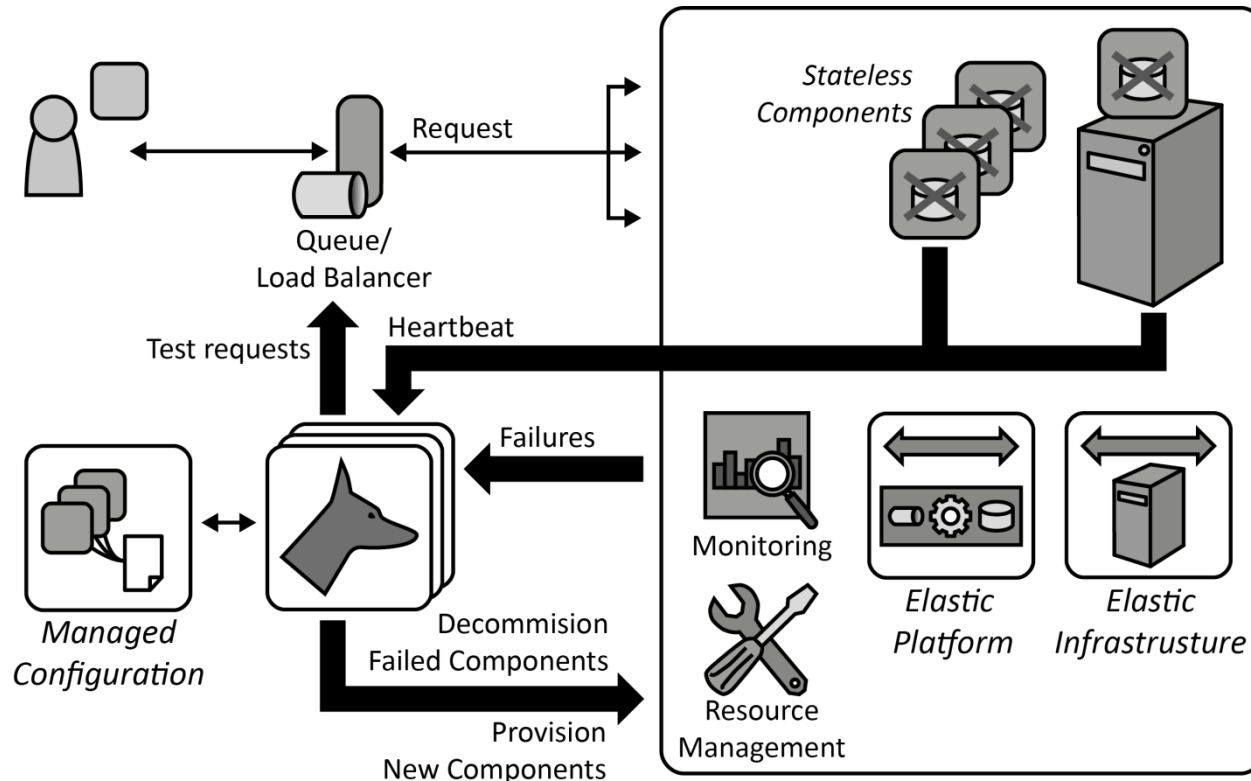
Integration Provider



Pattern Map: Cloud Management Patterns



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

Cloud Use Cases (Usage Scenarios)

- **Make data available to large communities**
- **Perform computation-intensive activities**
 - Peaky systems
- **One-time usage**
- **Support for short-term projects**
 - Low-risk startup/scaled agile
- **Make applications available to mobile users**
- **General-purpose software**
 - Email, build server running test cases
- **Backup**
- **Disaster recovery**

Reference: G. Lewis, Architectural Implications of Cloud Computing (SEI SATURN 2013 tutorial)

- **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)?

Agenda

■ Cloud computing fundamentals

- Definitions
- Patterns
- Usage scenarios
- Risks and inhibitors

■ Design options at selected PaaS providers

- Decisions required
- Decision criteria
- Good design practices

■ Architecture design for cloud

- IDEAL application properties
- Pitfalls to avoid
- Cloud readiness assessment and architectural refactoring

Goal of CDAR lab:

- **Develop criteria for public PaaS comparison/evaluations**
- **Harvest architectural knowledge from gained experience**

CDAR approach:

- **Develop new applications and partially reengineer existing ones – and deploy them to selected cloud offerings (in varying configurations)**
 - Four public/polyglot PaaS clouds investigated (so far)
 - Eight applications deployed and run (so far)
 - JS2E and JEE enterprise applications (of varying complexity)
 - Servlets, Spring MVC, ActiveMQ, WSDL/SOAP, JDBC, MongoDB, ...
 - Probe applications, e.g. system self information (diagnostics), socket connections

CDAR: Domain-Driven Design Sample Application (Experiment)

■ Comprehensive Spring application ([SourceForge](#))

- Common Patterns and APIs
- Unchanged since 2009
- Runs fine in standalone Tomcat and JBoss
- Only runs partially in one public PaaS cloud:

DDDAug26 ▾
Owner: Created: 2013 Aug 26 Status: active ID: zio/dddaug26
Location: <http://dddaug26.zio.cloudbees.net>

Development Operations Logs Configuration

Cloud Environment

Upgrade your RUN@cloud subscription to enable additional features such as: larger 24x7 application containers, CNAME aliases, clustering and automatic scaling.
[Upgrade subscription...](#)

Application Container
Select a ClickStack for your application to run in.
Tomcat 6

Select the container size to provision for instances of your application.
 Free (5 app limit!)
Limited application instance, 128MB RAM, throttled concurrency, automatic hibernation. Free

Redundancy and Scale
Choose your application's cluster model (ensure your application is cluster friendly).
Single Instance

```
java.lang.IllegalArgumentException: setAttribute: Non-serializable attribute se.citerus.dddsample.interfaces.tracking.CargoTrackingController.FORM.trackCommand
org.apache.catalina.session.StandardSession.setAttribute(StandardSession.java:1351)
org.apache.catalina.session.StandardSession.setAttribute(StandardSession.java:1312)
org.apache.catalina.session.StandardSessionFacade.setAttribute(StandardSessionFacade.java:130)
org.springframework.web.servlet.mvc.AbstractFormController.showForm(AbstractFormController.java:566)
org.springframework.web.servlet.mvc.SimpleFormController.showForm(SimpleFormController.java:198)
org.springframework.web.servlet.mvc.SimpleFormController.showForm(SimpleFormController.java:175)
org.springframework.web.servlet.mvc.AbstractFormController.showNewForm(AbstractFormController.java:338)
org.springframework.web.servlet.mvc.AbstractFormController.handleRequestInternal(AbstractFormController.java:278)
org.springframework.web.servlet.mvc.AbstractController.handleRequest(AbstractController.java:153)
org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter.handle(SimpleControllerHandlerAdapter.java:48)
org.springframework.web.servlet.DispatcherServlet.doDispatch(DispatcherServlet.java:875)
org.springframework.web.servlet.DispatcherServlet.doService(DispatcherServlet.java:807)
org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java:571)
org.springframework.web.servlet.FrameworkServlet.doGet(FrameworkServlet.java:501)
javax.servlet.http.HttpServlet.service(HttpServlet.java:617)
javax.servlet.http.HttpServlet.service(HttpServlet.java:717)
com.opensymphony.module.sitemesh.filter.PageFilter.parsePage(PageFilter.java:119)
com.opensymphony.module.sitemesh.filter.PageFilter.doFilter(PageFilter.java:55)
```

note The full stack trace of the root cause is available in the Apache Tomcat/6.0.35 logs.

CDAR: PaaS Platform 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?

CDAR: PaaS Platform 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

CDAR: Architectural Issues Identified (Decisions Required)

■ Application server vs. native cloud PaaS services

- E.g. one of the evaluated PaaS providers only supports a subset of JEE
- JEE future is discussed controversially, see e.g. <https://devcenter.heroku.com/articles/intro-for-java-developers>

■ External interfaces, ports (operational model/node topology)

- E.g. Identity/access manager for role-based access control (if any)
- E.g. RMI communication (registry)
- E.g. Messaging with ActiveMQ (or other MOM provider, e.g. cloud-internal)

■ Virtual Private Network (VPN) between Web application and database server

- E.g. Amazon <http://aws.amazon.com/de/vpc/>

■ URI design

- Virtual host and relative URIs

CDAR: More Architectural Issues Identified (Decisions Required)

■ Integration in a hybrid cloud

- Federated identity management
- Data replication, data migration

■ Approach to systems and service management

- Custom systems management and/or usage of provider capabilities?
 - Tradeoff: control and provider independence vs. effort, level of detail
- CSAR design (if TOSCA is used)

■ Approach to billing (if any), integration of payment services

- Billing of cloud solution to client (IaaS, PaaS, SaaS)
- Billing of cloud usage (SaaS, PaaS, IaaS)
 - Including initial and continuous data transfer (batch/FTP or RESTful HTTP)

CDAR: Deployment Considerations

■ Build Procedures and Deployment Process

- Traditional staging (unit test/integration test/acceptance test/production?)
 - Some cloud users report to deploy to production directly (highly agile approach)

■ In public PaaS context:

- Who tests service updates pushed into production instance via GitHub?
- Is it ok to host an RDBMS on a public IaaS offering and expose JDBC port on the Internet (e.g. port 3306)?
- What if a public PaaS provider uses another cloud provider for IaaS?
- Are the HTTP request/session timeout settings defined by cloud provider acceptable? Can they be overwritten?
- Same for other connection types, e.g. JDBC

502 Bad Gateway

nginx/1.4.1

Good Cloud Design Practices (from CDAR Lab)

- **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)**

Agenda

■ Cloud computing fundamentals

- Definitions
- Patterns

■ Design options at selected PaaS providers

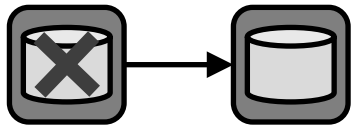
- Decisions required
- Decision criteria
- Good design practices

■ Architecture design for cloud

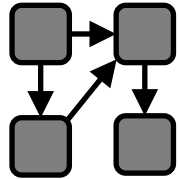
- IDEAL application properties
- Pitfalls to avoid
- Cloud readiness assessment and architectural refactoring

IDEAL Cloud Application Properties

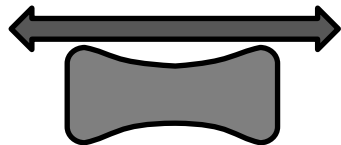
Reference: Cloud Computing Patterns, Springer 2014, <http://cloudcomputingpatterns.org/>



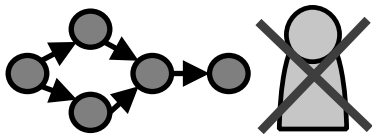
Isolated State: most of the application is *stateless* with respect to:
Session State: state of the communication with the application
Application State: data handled by the application



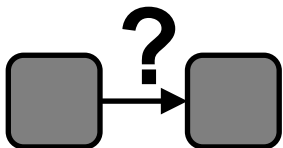
Distribution: applications are decomposed to...
... use multiple cloud resources
... support the fact that clouds are large globally distributed systems



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

- **J. Varia from Amazon (Reference: <http://aws.amazon.com/whitepapers/>)**
 - **“Design for failure and nothing will fail”**
 1. Have a coherent backup and restore strategy for your data and automate it
 2. Build process threads that resume on reboot
 3. Allow the state of the system to re-sync by reloading messages from queues
 4. Keep pre-configured and pre-optimized virtual images to support (2) and (3) on launch/boot
 5. Avoid in-memory sessions or stateful user context, move that to data stores.
 - **Decouple your components**
 - **Implement elasticity**
 - Automate your infrastructure
 - Bootstrap your instances
 - **Think parallel**
 - **Keep dynamic data closer to the compute and static data closer to the end-user”**

■ **ARC context and project goal**

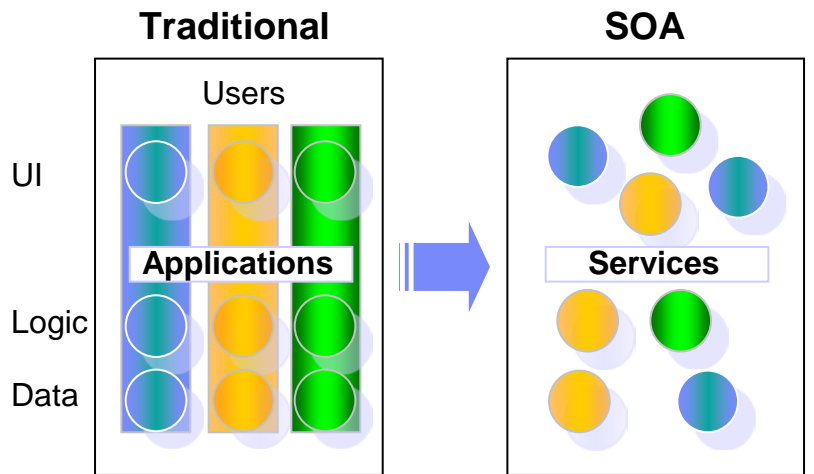
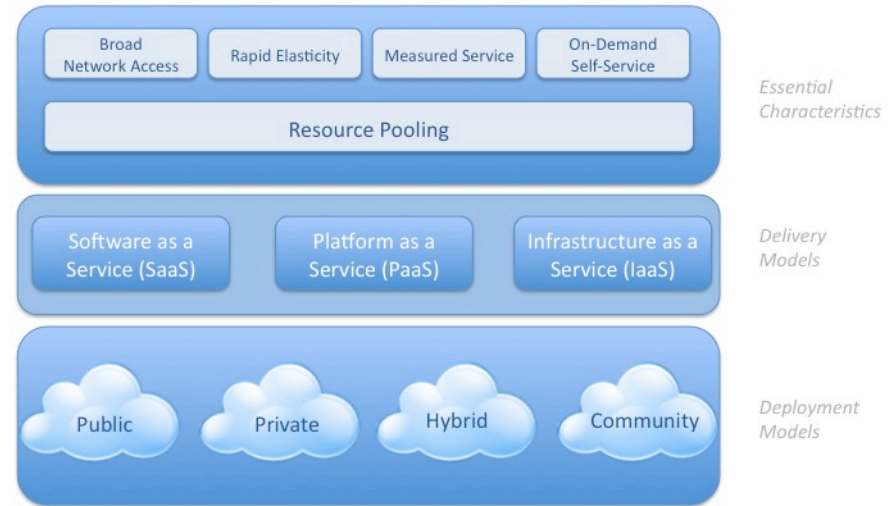
- Architectural knowledge about cloud design still lacking
 - Cloud provider, cloud user
- Migration to cloud/legacy system transformation is a large-scale refactoring
- Code refactoring is a very popular practice in development (agile community)
 - Supported by Eclipse and other IDEs

 **Create and manage architectural knowledge about cloud refactorings**

ARC: From Traditional Layer-Tier Architectures to Cloud Services



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



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

Basket of Services

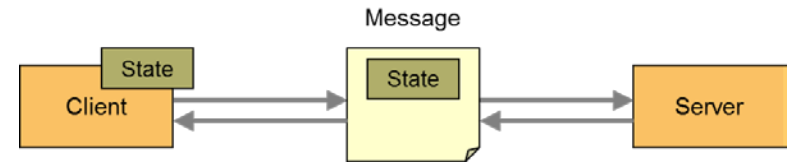
Decision Required: Session State Management

- **An example of a recurring design issue when moving a Web application to a cloud is *session state management*.**
- **According to [Patterns of Enterprise Application Architecture](#) by M. Fowler, the three top-level design options are:**
 - Client Session State (HTML/HTTP: cookie, hidden field, URL rewrite)
 - Server Session State (JEE: HTTP session object)
 - Database Session State (JEE: SQL DB via JDBC)
- **This decision has to be made/revisited when taking any Web-based business application to the cloud; outcomes may vary, but issues and options stay the same (i.e., they recur).**
 - Architects remain master of their project's destiny, but their decisions are backed by the recommendations given by the community.

Cloud Example: Refactor State Management Design

■ Client Session State

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



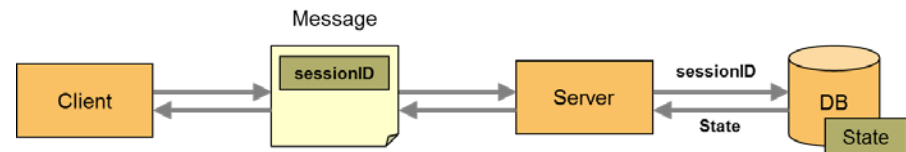
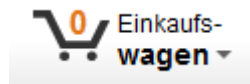
■ Server Session State

- 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	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/eaCatalog/>

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/eaCatalog/>

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/eaCatalog/>

Patterns to Improve Startup Times

- **Consider the Lazy Load pattern from PoEAA**
 - <http://www.martinfowler.com/eaCatalog/lazyLoad.html>
- **Not recommended: Eager Load – anti pattern**
 - E.g. avoid pre-loading caches on startup (a variant of Eager Load)
 - <http://www.thedwick.com/2010/06/performance-anti-pattern-pre-loading-caches-on-startup/>
- **Similar considerations/tradeoffs apply for initialization of other application resources**
 - Connection pools, factories
 - Bean pools, system transactions
 - User interface configuration (selection dialogs, validation rules)

Initial Ideas for Content of Arch. Ref. (AR) Catalog for Cloud

■ Change cloud computing pattern

- E.g. from server state to database state management to support horizontal scaling (sharding)
- E.g. from normalized to partitioned/replicated master data to support NoSQL storage of transactional data
- E.g. from flat rate to usage-based billing to support elasticity in a cost-efficient manner

■ See separate OOP session for more about architectural refactoring:

Architectural Refactoring - agile Umsetzung von Modernisierungsentscheidungen

Datum: 05.02.2014 Uhrzeit: 11:00 - 11:45 Vortrag: Mi 6.2



Architectural Refactoring: [Name]	
Context (viewpoint, refinement level): <ul style="list-style-type: none">• [...]	Quality attributes and stories (forces): <ul style="list-style-type: none">• [...]
Smell (refactoring driver): <ul style="list-style-type: none">• [...]	
Architectural decision(s) to be revisited: <ul style="list-style-type: none">• [...]	
Refactoring (solution sketch/evolution outline): <ul style="list-style-type: none">• [...]	
Affected components and connectors (if modelled explicitly): <ul style="list-style-type: none">• [...]	
Execution tasks (in agile planning tool and/or full-fledged design method): <ul style="list-style-type: none">• [...]	

Towards a Cloud Domain Refactoring Catalog (Preview)

Category	Refactorings		
IaaS	Virtualize Server	Virtualize Storage	Virtualize Network
IaaS, PaaS	Swap Cloud Provider	Change Operating System	Open Port
PaaS	“De-SQL”	“BASEify” (remove “ACID”)	Replace DBMS
PaaS	Change Messaging QoS	Upgrade Queue Endpoint(s)	Swap Messaging Provider
SaaS/application	Increase Concurrency	Add Cache	Precompute Results
SaaS/application	(CCP book, CBDI-SAE)	(all Stal refactorings)	(PoEAA/Fowler patterns)
Scalability	Change Strategy (Scale Up vs. Scale Out)	Replace Own Cache with Provider Capability	Add Cloud Resource (xaaS)
Performance	Add Lazy Loading	Move State to Database	
Communication	Change Message Exchange Pattern	Replace Transport Protocol	Change Protocol Provider
User management	Swap IAM Provider	Replicate Credential Store	Federate Identities
Service/deployment model changes	Move Workload to Cloud (use XaaS)	Privatize Deployment, Publicize Deployment	Merge Deployments (Use Hybrid Cloud)

Application Migration (including Database/Data Access Layer)

How to Adapt Applications for the Cloud Environment Challenges and Solutions in Migrating Applications to the Cloud

Vasilios Andrikopoulos, Tobias Binz, Frank Leymann, Steve Strauch

Decision Support for the Migration of the Application Database Layer to the Cloud

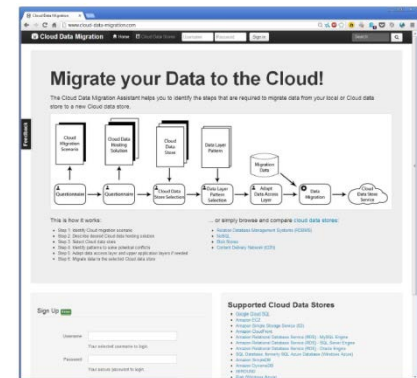
Steve Strauch, Vasilios Andrikopoulos, Thomas Bachmann, Dimka Karastoyanova, Stephan Passow, and Karolina Vukojevic-Haupt

■ Research project(s) at University of Stuttgart

- Author's copies of publications can be found via [this page](#) (start with the above two)

■ Online tool assisting with planning and executing migrations to the cloud (research prototype):

- Interview: current state, desired state
- Output: migration advice and automation snippets
- <http://www.cloud-data-migration.com/>



This is how it works:

- Step 1. Identify Cloud migration scenario
- Step 2. Describe desired Cloud data hosting solution
- Step 3. Select Cloud data store
- Step 4. Identify patterns to solve potential conflicts
- Step 5. Adapt data access layer and upper application layers if needed
- Step 6. Migrate data to the selected Cloud data store

ARC: 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)

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)
- **Cloud Computing Patterns sind in der Literatur verfügbar – Beispiele:**
 - At-Least-Once Delivery
 - Map-Reduce
 - Key-Value Storage und weitere NoSQL-Datenbanktypen
- **Cloud Design ist Software-Architektur-Design**
 - Viel Management-Bedarf, Security-Fragestellungen
- **HSR-Projekte: CDAR Lab, ARC, IFS [Software Health Check](#) (for Cloud)**

■ **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

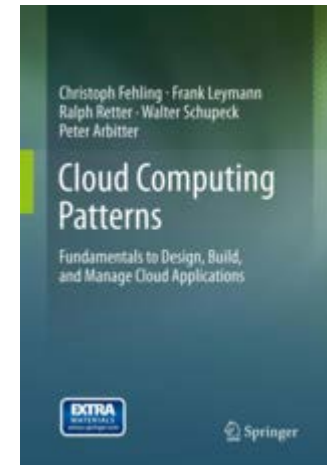
- <http://www.ifs.hsr.ch/Olaf-Zimmermann.11623.0.html?&L=4>
and ozimmerm@hsr.ch

■ 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>

InfoWorld Home / Blogs / Cloud Computing



Subscribe to Feed | Contact | Blogger Bio

When it makes sense to become a cloud provider

SEPTEMBER 03, 2013

Although most companies shouldn't consider becoming public cloud providers, it makes good sense in certain situations

1 Comment

Tags: Cloud computing, IT Management

[Read more »](#)